Potential-Guided Swarm Navigation for Lunar Microrovers

MSc Thesis

Gijs Zijderveld

TU Delft

Potential-Guided Swarm Navigation for Lunar Microrovers

by

Gijs Zijderveld

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday May 26, 2026 at 15:00.

Student number:	5306728
Project duration:	2025 – 2026
Thesis committee:	Prof. Dr. Javier Alonso-Mora TU Delft, supervisor
	Dr. Raj Thilak Rajan TU Delft, daily supervisor
	Dr. Laura Ferranti TU Delft, committee member

Cover:	Lunar Zebro rover render on a lunar-inspired surface.
Style:	TU Delft report style, adapted for MSc Robotics thesis.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

Abstract

Reliable autonomous navigation is a central requirement for future lunar rover swarms operating in cluttered, partially known terrain. Small rover swarms offer advantages in redundancy, coverage, and mission flexibility, but their navigation problem is more demanding than single-rover path planning: the system must avoid hazards, handle incomplete obstacle knowledge, prevent inter-agent interference, and select routes feasible for the group footprint. This thesis investigates how the Robust Artificial Potential Field (RAPF) framework can be extended toward reliable goal reachability for lunar rover swarms in obstacle-dense environments.

The work first improves the single-agent RAPF planner through systematic hyperparameter optimization, midpoint collision checking, and partial-trajectory repair. The resulting Improved RAPF (I-RAPF) planner reduces physically invalid path segments and avoids unnecessary full-path recomputation after local-minimum events. In controlled full-map experiments, I-RAPF achieves near-perfect reachability while reducing planning effort compared with the tuned RAPF baseline. In a sensing-based navigation loop, where obstacles are discovered incrementally, I-RAPF achieves an overall success rate of 99.84%, compared with 97.96% for Tuned RAPF, while also requiring fewer replanning events, lower planning effort, and shorter executed paths.

The thesis then extends I-RAPF to the swarm setting through a layered Coordinator-Guided Swarm architecture. The proposed method combines a formation-aware I-RAPF planner, a tunnel-flocking coordination layer, and a motion primitive execution layer. The formation-aware planner generates a shared path and formation-radius profile, representing both the desired route and the lateral space available to the swarm. The tunnel-flocking layer converts this route into local desired velocities, while the motion primitive layer maps these velocities to executable non-holonomic rover actions under local safety constraints.

Simulation benchmarks compare the Coordinator-Guided Swarm stack against a Self-Planned Swarm baseline, in which all agents use I-RAPF independently while sharing obstacle detections. The results show that shared sensing alone is not sufficient for robust collective traversal: independently planned routes can still cause incomplete collective arrival, traffic-like interference, and inconsistent route choices. Coordinator-guided formation-aware route management improves full-swarm convergence by providing a common traversal structure. Preliminary real-rover validation on the Lunar Zebro platform further demonstrates integration of I-RAPF into the physical navigation stack and identifies practical deployment limitations related to localization quality and experimental logging.

Overall, the thesis shows that RAPF can be strengthened into a reliable single-agent planner and extended to swarm navigation when route-level formation feasibility is combined with local coordination and primitive-based execution. The results support the central conclusion that robust lunar rover swarm navigation requires both shared obstacle awareness and shared route organization that accounts for the spatial requirements of the swarm.

Acknowledgements

I would like to thank everyone who supported me during the work on this thesis.

First of all, I would like to thank my daily supervisor, Raj Thilak Rajan, for his guidance, feedback, and encouragement throughout the project. Our weekly meetings helped me to keep direction in a project that touched many different topics, from path planning and swarm coordination to simulation and real-rover testing. His direct feedback on my writing, research direction, and technical choices was extremely valuable. I am also grateful for the opportunities he created to discuss and present my work outside the thesis itself, including events such as Highlights Delft. These moments helped me to better explain my research and place it in a broader context.

I would also like to thank my supervisor, Javier Alonso-Mora, for his valuable feedback and for sharing his deep understanding of planning, decision making, and autonomous mobile robotics. His input helped me to think more carefully about the structure and motivation of the proposed methods. I also appreciated the monthly AMR meetings, where students from the research group could discuss the problems they were facing, give feedback to each other, and brainstorm possible solutions. These meetings were a valuable source of ideas and helped me to view my work from different perspectives.

I am grateful to Laura Ferranti for being part of my thesis committee and for taking the time to review and evaluate this work.

I would like to thank the Lunar Zebro research group for providing the setting in which this thesis could take shape. Working on navigation methods for lunar microrovers has been a challenging and rewarding experience. It gave me the opportunity to work on a topic that connects path planning, swarm coordination, simulation, and real-world robotic experiments.

I am also thankful to everyone who helped with practical discussions, experimental setup, software issues, and testing. A thesis like this is never only the result of the final text or the final experiments; it also depends on many smaller conversations, debugging sessions, and pieces of advice along the way.

Finally, I would like to thank my family and friends for their support throughout my studies. Their encouragement, patience, and occasional distraction when needed made the process much easier and more enjoyable.

This thesis concludes my MSc Robotics programme at Delft University of Technology. I am grateful for the people I have met, the things I have learned, and the opportunity to contribute to a topic that I find genuinely exciting.

Contents

Nomenclature

Sets and Indices

$(\cdot)_i$ Subscript indicating a quantity associated with agent i

$\hat{\mathcal{S}}_{i,k}(u)$ Predicted swept set of agent i under primitive u

$\mathcal{A}$ Finite set of elementary rover actions, such as forward motion, a fixed left turn, and a fixed right turn

$\mathcal{B}_k$ Set of formation-augmented candidate pairs at planning step k, containing candidate positions and their largest feasible formation radius

$\mathcal{C}_k$ Set of candidate target positions, or bacteria, at planning step k, $\mathcal{C}_k \subset \mathbb{R}^{2\times 1}$

$\mathcal{D}$ Set of obstacle detections returned by the current sensing action

$\mathcal{D}_i$ Set of most recent obstacle detections made by agent i during the current sensing action

$\mathcal{I}$ Set of agent indices, $\mathcal{I} = \{1, \ldots, N\}$

$\mathcal{N}_i$ Set of neighbouring agents within communication range of agent i at the current navigation step

$\mathcal{O}_{\mathrm{ACT},i}$ Active obstacle set currently known to agent i, updated through local sensing and communicated detections

$\mathcal{O}_{\mathrm{LOC},i}$ Local obstacle set considered by agent i during execution, with $\mathcal{O}_{\mathrm{LOC},i} \subseteq \mathcal{O}_{\mathrm{ACT},i}$

$\mathcal{O}_{ACT}$ Active obstacle set used by the planner at the current navigation step

$\mathcal{O}_{AO}$ Set of artificial obstacles generated to escape local minima

$\mathcal{O}_{STAT}$ Set of static environmental obstacles such as rocks and craters

$\mathcal{P}$ Planned path represented as an ordered waypoint sequence, $\mathcal{P} = \{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_M\}$, where $\mathbf{w}_k \in \mathbb{R}^{2\times 1}$

$\mathcal{R}_{\mathcal{P}}$ Formation-radius profile associated with the planned path $\mathcal{P}$, $\mathcal{R}_{\mathcal{P}} = \{r_{\mathrm{form},0}, \ldots, r_{\mathrm{form},M}\}$

$\mathcal{R}_{\mathrm{form}}$ Ordered set of candidate formation radii, $\mathcal{R}_{\mathrm{form}} = \{r_{\mathrm{form}}^{(1)}, \ldots, r_{\mathrm{form}}^{(n_r)}\}$

$\mathcal{U}$ Finite set of available motion primitives

$\mathcal{U}_{i,k}^{\mathrm{safe}}$ Set of motion primitives that satisfy obstacle and neighbour feasibility constraints for agent i at navigation step k

$\mathcal{U}_{\text{arc}}$ Subset of motion primitives that combine translation and heading change

$\mathcal{U}_{\text{fwd}}$ Subset of motion primitives that primarily move the robot forward

$\mathcal{U}_{\text{rec}}$ Subset of optional recovery primitives, such as reverse or reverse-arc motions

$\mathcal{U}_{\text{turn}}$ Subset of motion primitives that primarily rotate the robot

i Agent index, with $i \in \mathcal{I}$

Kinematics and State

$\Delta\mathbf{p}_k$ Recorded planar displacement of the rover between two consecutive localization samples, $\Delta\mathbf{p}_k = [x_{k+1} - x_k,\ y_{k+1} - y_k]^T$

$\Delta\hat{\mathbf{p}}_{i,k}(u)$ Predicted planar displacement of agent i under primitive u, $\Delta\hat{\mathbf{p}}_{i,k}(u) \in \mathbb{R}^{2\times1}$

ℓ Waypoint or formation-radius index along a planned path

$\hat{\mathbf{x}}_{i,k+1}(u)$ Predicted next pose of agent i after applying primitive u, $\hat{\mathbf{x}}_{i,k+1}(u) \in \mathbb{R}^{3\times1}$

$\mathbf{c}_k$ Candidate waypoint, or bacterium, generated at planning step k, $\mathbf{c}_k \in \mathbb{R}^{2\times1}$

$\mathbf{e}_{h,k}$ Unit vector in the recorded rover heading direction at localization sample k, $\mathbf{e}_{h,k} = [\cos(h_k),\ \sin(h_k)]^T$

$\mathbf{g}$ Global target goal position vector, $\mathbf{g} \in \mathbb{R}^{2\times1}$

$\mathbf{m}_k$ Midpoint of the segment between the current position and candidate waypoint $\mathbf{c}_k$, $\mathbf{m}_k \in \mathbb{R}^{2\times1}$

$\mathbf{n}_{i,k}$ Outward unit normal from the path centreline to agent i, $\mathbf{n}_{i,k} \in \mathbb{R}^{2\times1}$

$\mathbf{p}_k^{\text{ref}}$ Reference position of the shared route at planning step k, $\mathbf{p}_k^{\text{ref}} \in \mathbb{R}^{2\times1}$

$\mathbf{p}_k$ Position vector of the rover at planning step k, $\mathbf{p}_k \in \mathbb{R}^{2\times1}$

$\mathbf{p}_o$ Position vector of obstacle o, $\mathbf{p}_o \in \mathbb{R}^{2\times1}$

$\mathbf{p}_{\text{stuck}}$ Stagnation position at which an artificial obstacle is inserted, $\mathbf{p}_{\text{stuck}} \in \mathbb{R}^{2\times1}$

$\mathbf{p}_{i,k}$ Position vector of agent i at navigation step k, $\mathbf{p}_{i,k} \in \mathbb{R}^{2\times1}$

$\mathbf{u}_k$ Displacement vector applied at step k, $\mathbf{u}_k \in \mathbb{R}^{2\times1}$

$\mathbf{v}_{i,k}^{\text{des}}$ Desired velocity vector generated for agent i by the tunnel-flocking layer, $\mathbf{v}_{i,k}^{\text{des}} \in \mathbb{R}^{2\times1}$

$\mathbf{v}_{i,k}^{\text{eff}}(u)$ Effective velocity of agent i at navigation step k induced by primitive u, $\mathbf{v}_{i,k}^{\text{eff}}(u) \in \mathbb{R}^{2\times1}$

$\mathbf{v}_{i,k}$ Current velocity vector of agent i at navigation step k, $\mathbf{v}_{i,k} \in \mathbb{R}^{2\times1}$

$\mathbf{w}_{i,k}^{\text{ref}}$ Reference point or look-ahead point on $\mathcal{P}$ used by agent i at navigation step k, $\mathbf{w}_{i,k}^{\text{ref}} \in \mathbb{R}^{2\times1}$

$\mathbf{w}_\ell$ Waypoint ℓ in the planned path, with $\mathbf{w}_\ell \in \mathbb{R}^{2\times1}$

$\mathbf{x}_{i,k}$ Pose vector of agent i at navigation step k, $\mathbf{x}_{i,k} = [x_{i,k}, y_{i,k}, \theta_{i,k}]^T \in \mathbb{R}^{3\times 1}$

a Elementary rover action, with $a \in \mathcal{A}$

$d_{i,k}^{\text{tube}}$ Perpendicular distance from agent i to the relevant path segment at navigation step k

$d_{\text{fwd},k}$ Heading-projected forward displacement component between localization samples k and $k+1$

d_{goal} Final Euclidean distance between the rover and the goal at the end of a failed simulation

e Number of internal planning iterations executed during one episode

h_k Recorded rover heading at localization sample k, expressed in radians

k Discrete planning or navigation step index, $k \in \mathbb{N}_0$

l_{hp} Heading-projected executed-distance estimate used in the real-rover validation

l_{path} Executed path length of one episode

n_u Number of simulation steps covered by primitive u

$N_{\text{agent},i}$ Accumulated number of inter-agent collisions in rollout i

n_{ao} Number of artificial obstacles inserted during one episode

$n_{\text{collision}}$ Number of collisions during one episode

$N_{\text{obs},i}$ Accumulated number of obstacle collisions in rollout i

$n_{\text{reached},i}$ Number of agents that reached the goal region in rollout i

n_{restart} Number of restart or repair recovery events during one episode

$N_{\text{steps},i}$ Duration of rollout i in simulation steps

q Index of a candidate formation radius in $\mathcal{R}_{\text{form}}$

$r_{i,k}^{\text{tube}}$ Local admissible tube radius obtained from $\mathcal{R}_{\mathcal{P}}$ for agent i at navigation step k

r_o Radius of obstacle o

r_{form} Formation radius used to represent the admissible swarm footprint around the shared route

$r_{\text{overspeed},i}$ Fraction of raw desired-velocity commands exceeding the maximum velocity before clamping in rollout i

$s_{\text{osc},i}$ Mean oscillation score computed from changes in executed velocity in rollout i

T_u Execution duration of primitive u

$u_{i,k}$ Motion primitive selected by agent i at navigation step k, with $u_{i,k} \in \mathcal{U}$

Evaluation Functions

$\chi_{\text{rec}}(u)$ Indicator or penalty term for recovery primitives

$\Delta s_i(u)$ Progress along the path direction produced by primitive u for agent i

$\gamma_{\text{sched}}(\cdot)$ Gain-scheduling function that selects controller gains from the local tube radius

$\mathbb{I}_{\text{success},i}$ Success indicator for rollout i, equal to one if all agents reach the goal region and zero otherwise

$\psi_{\text{obs}}(\cdot)$ Distance-dependent obstacle repulsion function

$\psi_{\text{sep}}(\cdot)$ Distance-dependent inter-agent separation function

$c_{\text{collision}}$ Cost term associated with collisions during execution

c_{repair} Cost term associated with recovery from local minima

c_{thinking} Cost term associated with computational planning effort

$e_\theta(u)$ Heading-alignment error term used in primitive scoring

$F_R(D)$ Cumulative fractional area covered by rocks larger than diameter D

$f_{\text{prim}}(\mathbf{x}_{i,k}, u)$ Primitive transition model predicting the next pose after applying primitive u

$j(\mathbf{c})$ Scalar navigation cost evaluated at candidate point $\mathbf{c}$

j_a Attractive potential function component towards the goal

j_o Repulsive potential function component from obstacles

$j_{\text{prim}}(u)$ Motion primitive selection score for primitive u

$j_{\text{rapf}}(\mathbf{c})$ Local RAPF successor score evaluated at candidate position $\mathbf{c}$

j_{trial} Average optimization objective value of one hyperparameter trial

s_i Scalar rollout or scenario score for case i during controller or hyperparameter evaluation

s_i^{succ} Scenario score for a successful navigation episode in environment i

Parameters

Δt Simulation time step used in the primitive execution model

A_a, M_a Constants defining the depth and width of the attractive potential well

A_o, M_o Constants determining the magnitude and decay of repulsive influence

D_{dead} Deadband threshold below which heading-projected displacement components are ignored

$H_{\text{influence}}$ Influence horizon of an inserted artificial obstacle, defined as $H_{\text{influence}} = R_{AO} + R_u$

K_R Constant representing rock abundance percentage on the lunar surface

N Total number of agents in the swarm

N_B Constant number of candidate bacteria sampled per planning step

Q_R Empirically fitted coefficient for rock size-frequency distribution

R_B Constant magnitude of the displacement vector, or bacteria step size

R_l, R_u Lower safety radius and upper influence radius for repulsive potentials

R_{COMM} Communication radius within which agents exchange detections and state information

R_{ROBOT} Robot body radius used for local primitive feasibility checks

R_{SENSE} Sensing radius within which an agent detects nearby obstacles

R_{AO} Constant radius of an artificial obstacle inserted to escape local minima

R_{GOAL} Threshold radius defining the goal region

R_{SAFE} Minimum required safety distance between agents or obstacles

V_{MAX} Maximum commanded speed

Chapter 1

Introduction

The exploration of the lunar surface presents a significant opportunity for scientific discovery and resource utilization. At the same time, it poses severe operational challenges for robotic systems. The lunar surface is unstructured, obstacle-dense, and difficult to predict at the scale relevant for small mobile robots. Rocks, crater rims, and irregular terrain can all obstruct progress or trap a vehicle in locally unfavorable configurations. Traditional large-scale rovers are powerful, but they are also costly and represent a single point of failure. In response, recent research efforts, including TU Delft's Lunar Zebro project, have increasingly turned toward swarms of smaller, more affordable, and fault-tolerant micro-rovers. Such swarms offer clear potential advantages: they can cover more ground, gather data in parallel, and tolerate the loss of individual agents without ending the mission.

These advantages, however, depend critically on the swarm's ability to navigate autonomously. The challenge is not only to move a single rover safely through cluttered terrain, but to do so with multiple robots that must avoid collisions, maintain useful spatial organization, and still make collective progress toward a shared goal. In other words, the navigation problem is both geometric and cooperative: the terrain constrains where the robots can go, while the swarm itself introduces additional spatial and coordination requirements.

A further complication is that this problem must be solved under partial observability. A lunar rover swarm does not move through a fully known environment with a complete map available in advance; instead, obstacle information is revealed incrementally through local sensing during execution. Navigation therefore cannot be treated purely as a one-shot path generation problem. As newly perceived terrain invalidates previously feasible routes, the system must repeatedly validate, repair, or replace its current plan. In such settings, graph-search and sampling-based planners such as A* [1], D* Lite [2], and RRT* [3] remain important alternatives, but repeated route repair becomes a central practical concern.

Artificial Potential Field (APF) methods provide an attractive starting point for this problem. Since the classical formulation introduced by Khatib [4], APF has remained appealing because it is reactive, computationally lightweight, and naturally suited to local obstacle avoidance. These properties are especially relevant for low-power robotic systems operating under strict onboard computational limits, as is often the case in planetary exploration. However, the same literature also makes clear that classical APF suffers from important reachability limitations. As shown by Matoui et al. [5] and Puriyanto et al. [6], the superposition of attractive and repulsive influences can create local minima, oscillatory motion, and deadlock-like behavior that prevent reliable goal convergence in cluttered environments. Classical APF is therefore attractive from a computational perspective, but insufficient when reliable reachability is required.

A substantial body of work has sought to improve this situation for single-robot navigation. Existing APF extensions modify the field itself [6], adapt repulsion near obstacles [7], or introduce explicit local-minima escape mechanisms [8]. Of particular relevance to this thesis is the Robust Artificial Potential Field (RAPF) algorithm proposed by Manteaux et al. [9], which preserves the lightweight and reactive character of APF while improving single-agent robustness through bacterial sampling and artificial obstacle placement. RAPF therefore provides a compelling foundation for lunar rover navigation, but it was developed for the single-agent case.

The swarm case intensifies the same planning difficulty. If swarm size is incorporated directly into the planning problem, the planner must reason in a richer joint planning space than in the single-agent case, and the effective search burden can grow rapidly with the number of robots [10]. Under partial observability, where replanning may be required repeatedly as new terrain is sensed, this additional computational burden becomes even more consequential. This makes it particularly relevant to investigate whether a lightweight reactive planning principle can be extended from the single-agent case to the swarm setting.

Once multiple robots are introduced, the problem changes qualitatively. In multi-robot settings, failures no longer arise only from terrain-induced local minima, but also from interactions between the robots themselves. Prior work has explored several ways of extending APF-based navigation toward coordinated group behavior. Bao et al. [11] apply APF within a leader–follower framework for lunar-style swarm navigation, Adderson et al. [12] study decentralized formation adaptation in cluttered environments, and Abdi et al. [13] show how shared local-minima information can improve collective navigation efficiency. More broadly, flocking- and behavior-based coordination methods have demonstrated that local interaction rules can help robot groups move coherently while remaining decentralized [14, 15]. Together, these studies show that coordination and information sharing can mitigate some of the weaknesses of single-robot APF when moving to the swarm setting.

At the same time, they also suggest an important limitation. Most of these approaches strengthen coordination during execution, but they still tend to assume that a route feasible for one robot is, with sufficient coordination effort, also feasible for the swarm. In highly cluttered terrain, that assumption often breaks down. A path that is collision-free for an individual robot may still provide insufficient clearance for a group with a non-negligible spatial footprint. Small deviations during execution can then push agents into nearby obstacles, fragment the swarm, or trigger deadlock-like interactions between robots competing for limited free space.

This observation points to a deeper planning requirement. Reliable swarm navigation in cluttered environments cannot depend on reactive coordination alone; it also requires that the geometry of the swarm be considered during route generation itself. Formation-aware planning literature shows that route feasibility may depend directly on group footprint, formation deformation, and reconfiguration in constrained regions [16, 17]. This is consistent with recent formation-aware planning approaches, where the feasibility and quality of a route depend not only on where a single robot can pass, but also on how much space the group requires and how that spatial requirement changes along the path [18, 19]. For the present problem, this suggests that extending RAPF to the swarm domain is not merely a matter of adding local coordination rules on top of a single-agent planner. Instead, the planner itself must begin to reason about swarm footprint and clearance.

The central research question of this thesis is therefore:

How can the Robust Artificial Potential Field (RAPF) navigation framework be extended to enable reliable swarm navigation in highly cluttered environments with densely distributed lunar rocks and craters?

To address this question, this thesis develops the RAPF framework in three stages. First, the baseline single-agent RAPF planner is optimized and structurally improved, resulting in Improved RAPF (I-RAPF). These improvements target planner robustness, physical path validity, and computational efficiency. Second, I-RAPF is evaluated in a sensing-based navigation loop, where obstacle information is acquired incrementally and paths must be repeatedly validated and repaired during execution. Third, the planner is extended to the swarm setting through formation-aware route management, tunnel-flocking coordination, and motion primitive execution.

The resulting swarm navigation stack uses a coordinator-guided shared route to provide the group with a common traversal structure. Instead of relying only on agents sharing obstacle detections and planning independently, the proposed method plans a route with the spatial footprint of the swarm in mind. During execution, the tunnel-flocking layer distributes the agents along this route, while the motion primitive layer converts desired velocities into feasible rover actions. This layered design allows the lightweight and reactive character of RAPF to be preserved while addressing the additional coordination requirements introduced by swarm navigation.

The main contributions of this thesis are:

- The development of Improved RAPF (I-RAPF), a strengthened single-agent RAPF planner that combines hyperparameter optimization with midpoint collision checking and partial-trajectory repair.
- A sensing-based evaluation of I-RAPF under partial observability, where obstacle information is acquired online and the planner must repeatedly validate, repair, or replace the current route.
- A layered swarm navigation architecture that separates formation-aware route management, local coordination, motion primitive execution, and communication.
- A formation-aware extension of I-RAPF, denoted FA-I-RAPF, that evaluates candidate route points with respect to an effective swarm footprint and produces both a shared path P and a formation-radius profile R_P.
- The integration of FA-I-RAPF with a tunnel-flocking controller and a motion primitive execution layer, linking route-level formation feasibility to local swarm motion and executable non-holonomic rover actions.
- A comparative swarm benchmark between the proposed Coordinator-Guided Swarm stack (CG-Swarm) and a Self-Planned Swarm baseline (SP-Swarm), isolating the value of shared formation-aware route management beyond shared obstacle sensing alone.
- A preliminary real-rover validation of I-RAPF on the Lunar Zebro rover platform, demonstrating integration of the improved planner into the physical navigation stack while identifying practical deployment limitations.

1.1 Document preview

The remainder of this thesis is structured as follows. Chapter 2 introduces the background and foundations used throughout the thesis. It discusses classical Artificial Potential Field methods, the baseline RAPF formulation, the rover motion abstraction, and the lunar-inspired benchmark environment.

Chapter 3 develops the improved single-agent planner used in the remainder of the thesis. It first identifies limitations of the baseline RAPF formulation, then introduces the hyperparameter optimization procedure and the algorithmic improvements that lead to Improved RAPF (I-RAPF).

Chapter 4 evaluates I-RAPF in a sensing-based navigation loop. In this setting, the rover does not know the full obstacle map in advance, but incrementally updates its active obstacle set, validates the current path, and replans when newly detected obstacles invalidate the route.

Chapter 5 introduces the swarm navigation problem and motivates the need for formation-aware route planning. It compares candidate coordination strategies and argues that local coordination alone is insufficient when the planned route does not provide enough clearance for the spatial footprint of the swarm.

Chapter 6 presents the proposed layered swarm navigation method. This chapter introduces the formation-aware I-RAPF planner, the tunnel-flocking execution layer, and the motion primitive layer that converts desired swarm motion into executable rover actions.

Chapter 7 evaluates the complete swarm navigation stack in simulation. The proposed Coordinator-Guided Swarm stack (CG-Swarm) is compared against a Self-Planned Swarm baseline (SP-Swarm) to isolate the effect of shared formation-aware route management.

Chapter 8 presents a preliminary real-rover validation of I-RAPF on the Lunar Zebro rover platform. This chapter compares the improved planner with the existing RAPF implementation in an indoor obstacle setup and discusses practical deployment issues.

Finally, Chapter 9 discusses the main findings, answers the research question, summarizes the contributions, identifies limitations, outlines future work, and concludes the thesis.

Chapter 2

Background and Foundations

This chapter establishes the theoretical, motion-model, and environmental foundations used throughout this thesis. It first introduces the development of Artificial Potential Field (APF) methods, starting from classical reactive navigation and progressing toward the Robust Artificial Potential Field (RAPF) algorithm in Section 2.1. Section 2.2 then formulates the baseline single-agent RAPF planner used as the starting point for the later algorithmic improvements.

After the planning formulation is introduced, the chapter clarifies in Section 2.3 the rover motion model used in this work. The physical rover is not assumed to move holonomically in the plane; instead, its motion is constrained by heading and by a finite set of short discrete actions. This distinction is important because the planner reasons about candidate positions and navigation costs, while the robot can only realize motion through executable manoeuvres.

Finally, Section 2.4 defines the lunar-inspired environment model used for evaluation. This includes the obstacle representation, geological hazard distributions, and benchmark scenarios used to test algorithmic performance across increasing levels of environmental complexity.

2.1 From Classical APF to RAPF

Khatib [4] introduced the classical Artificial Potential Field (APF) method as a reactive navigation strategy in which a robot moves through a virtual force field. In this formulation, the goal generates an attractive influence, while obstacles generate repulsive influences. The robot motion is then determined by the superposition of these effects, which makes APF computationally lightweight and responsive to local environmental changes.

A standard attractive potential is given by

$$U_{\text{att}}(x) = \frac{1}{2} k_{\text{att}} \|x - x_g\|^2, \tag{2.1}$$

where x denotes the robot position, x_g the goal position, and k_{att} the attractive gain. Obstacles generate repulsive potentials that are active only within a bounded influence radius,

$$U_{\text{rep}}(x) = \begin{cases} \frac{1}{2} k_{\text{rep}} \left(\frac{1}{\|x - x_o\|} - \frac{1}{d_0} \right)^2, & \|x - x_o\| \le d_0, \\ 0, & \text{otherwise}, \end{cases} \tag{2.2}$$

where x_o denotes the obstacle position, k_{rep} the repulsive gain, and d_0 the obstacle influence

distance. The resulting forces are obtained from the negative gradients of these potentials,

$$F_{\text{att}}(x) = -\nabla U_{\text{att}}(x), \qquad F_{\text{rep}}(x) = -\nabla U_{\text{rep}}(x), \tag{2.3}$$

such that the total force acting on the robot is

$$F_{\text{total}}(x) = F_{\text{att}}(x) + \sum_i F_{\text{rep},i}(x). \tag{2.4}$$

In classical APF, this force is directly translated into the robot motion.

Despite its simplicity, classical APF is known to suffer from local minima and oscillatory behaviour. As shown by Matoui et al. [5] and Puriyanto et al. [6], the attractive and repulsive influences may balance at locations that are not the goal, causing the robot to stagnate or follow inefficient zigzagging trajectories in cluttered environments. These limitations motivated a large body of work aimed at preserving the lightweight character of APF while improving its robustness.

One line of work modifies the field formulation itself. Wu et al. [7] introduced adaptive scaling of repulsive forces, while Puriyanto et al. [6] explored modified attractive and repulsive potential functions. Peng et al. [20] and Diab et al. [21] further proposed time-varying and adaptive formulations that reshape the field online in order to reduce trapping and improve progress toward the goal. These approaches retain the reactive nature of APF, but alter the structure of the field to make local minima less likely.

A second line of work augments APF with explicit escape mechanisms. Park and Lee [22] and Hachour [23] proposed virtual-obstacle strategies that insert artificial repulsive sources when the robot becomes trapped. Yang et al. [24] and Lv et al. [25] introduced intermediate-goal methods that temporarily redirect the robot around problematic obstacle configurations. Li et al. [8] combined APF with wall-following behaviour, allowing the robot to bypass regions where the potential field alone is insufficient. Although these methods differ in implementation, they all supplement APF with a lightweight mechanism for escaping stagnation.

A particularly relevant development for the present thesis is the move from direct force following to bacteria-based local sampling. Rather than applying the field gradient directly to the robot, bacteria-based methods generate a set of candidate points around the current position and evaluate the navigation objective at each of them. Diab et al. [21] showed that this preserves the local and computationally efficient character of APF while turning the planner into a more structured local search process. This shift is especially important here, because it provides the conceptual bridge toward Robust Artificial Potential Field (RAPF).

Building on this idea, Manteaux et al. [9] proposed RAPF as a bacteria-based APF variant with a more explicit mechanism for handling local minima. Instead of relying on random exploratory motion, RAPF marks detected trap locations as artificial obstacles, thereby preventing the planner from repeatedly returning to them. In addition, RAPF modifies the bacteria generation process such that one candidate is always aligned with the robot–target direction, which reduces oscillatory behaviour when only a limited number of bacteria points is used. RAPF thus forms the endpoint of the progression considered in this section: from classical force-based APF, through lightweight field shaping and escape mechanisms, to a cost-based local sampling method with explicit trap avoidance.

2.2 Single-Agent RAPF Formulation

The previous section positioned Robust Artificial Potential Field (RAPF) as the endpoint of the progression from classical force-based APF toward cost-based local sampling with

explicit trap avoidance. This section now formalizes the baseline single-agent RAPF model used throughout this thesis. The formulation follows the RAPF method proposed by Manteaux et al. [9], while expressing it in the notation used in the remainder of this work.

For the single-agent planning formulation, the rover is modeled at the planner level as a point mass evolving in discrete time. The rover position at planning step $k \in \mathbb{N}_0$ is denoted by

$$\mathbf{p}_k = [x_k, y_k]^\top \in \mathbb{R}^{2\times 1}. \tag{2.5}$$

At each step, the rover applies a displacement

$$\mathbf{u}_k \in \mathbb{R}^{2\times 1}, \tag{2.6}$$

yielding the update

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \mathbf{u}_k, \tag{2.7}$$

with

$$\|\mathbf{u}_k\|_2 = R_B. \tag{2.8}$$

which represents the physical distance traversed in one planning cycle.

2.2.1 Candidate Set

Rather than directly applying a local force vector as in classical APF, RAPF evaluates a finite set of candidate positions around the current rover state. The candidate set is defined as

$$\mathcal{C}_k = \{\mathbf{c}_{1,k}, \ldots, \mathbf{c}_{M,k}\} \cup \{\mathbf{p}_k\}, \qquad \mathcal{C}_k \subset \mathbb{R}^{2\times 1}, \tag{2.9}$$

where $\mathbf{c}_{m,k}$ are candidate points sampled around $\mathbf{p}_k$.

In the baseline RAPF formulation, these candidate points correspond to the bacteria samples used to probe the local navigation landscape. Their role is to transform navigation from direct gradient following into a local cost-based selection process.

2.2.2 RAPF Navigation Cost

Following Manteaux et al. [9], RAPF evaluates a scalar navigation cost at each candidate point. The total navigation cost is defined as

$$j(\mathbf{c}) = j_a(\mathbf{c}, \mathbf{g}) + \sum_{o\in\mathcal{O}_{\text{act}}(t)} j_o(\mathbf{c}, o), \tag{2.10}$$

where $\mathbf{g} \in \mathbb{R}^{2\times 1}$ denotes the goal position and $\mathcal{O}_{\text{act}}(t)$ is the active obstacle set at planning step t.

Attractive Cost

The attractive cost is modeled as

$$j_a(\mathbf{c}, \mathbf{g}) = -A_a e^{-M_a\|\mathbf{c}-\mathbf{g}\|_2^2}, \tag{2.11}$$

where:

- A_a defines the depth of the attractive well;
- M_a controls the width of the basin of attraction.

Repulsive Cost

Let $d(\mathbf{c}, o) \in \mathbb{R}_{\geq 0}$ denote the Euclidean distance from candidate point $\mathbf{c}$ to the surface of obstacle o. The repulsive cost is defined as

$$j_o(\mathbf{c}, o) = \begin{cases} \infty, & d(\mathbf{c}, o) \leq R_l, \\ A_o e^{-M_o\, d(\mathbf{c},o)^2}, & R_l < d(\mathbf{c}, o) \leq R_u, \\ 0, & d(\mathbf{c}, o) > R_u, \end{cases} \tag{2.12}$$

where:

- R_l is the lower radius defining the hard safety boundary;
- R_u is the upper radius defining the obstacle influence range;
- A_o determines the maximum repulsive cost near the obstacle;
- M_o controls the decay of the repulsive influence.

This piecewise formulation creates two obstacle zones: a hard exclusion region that guarantees collision avoidance and a softer influence region that steers the rover away before collision becomes imminent.

2.2.3 Baseline RAPF Update Logic

At each planning step k, the rover evaluates the total navigation cost $j(\mathbf{c})$ for all candidates in $\mathcal{C}_k$ and selects the most favourable one

$$\mathbf{c}_k^* = \arg\min_{\mathbf{c} \in \mathcal{C}_k} j(\mathbf{c}). \tag{2.13}$$

If a valid descending move is found, such that $j(\mathbf{c}_k^*) < j(\mathbf{p}_k)$, the rover advances to that candidate:

$$\mathbf{p}_{k+1} = \mathbf{c}_k^*. \tag{2.14}$$

If no sustained descent can be achieved, the current position is interpreted as a local minimum. In the baseline RAPF formulation of Manteaux et al. [9], the planner responds by inserting an artificial obstacle at the current location and restarting the path construction from the initial state. This mechanism allows the planner to explicitly avoid revisiting previously detected trap regions.

RAPF therefore replaces direct force application by local cost-based candidate selection, while preserving the reactive and lightweight nature that motivates APF-based navigation in the first place. In the present thesis, this baseline single-agent RAPF formulation serves as the starting point for the later algorithmic improvements. In particular, the full-restart behaviour associated with local-minimum handling motivates the partial-repair extensions introduced in Chapter 3.

The equations above define the individual components of the planner. Algorithm 1 summarizes how these components are combined in the baseline single-agent RAPF loop used as the starting point in this thesis.

Algorithm 1 Baseline single-agent RAPF [9]

Input: $\mathbf{p}_0 \in \mathbb{R}^{2\times 1}$, $\mathbf{g} \in \mathbb{R}^{2\times 1}$, $\mathcal{O}_{\text{STAT}}$
Output: Planned path $\mathcal{P} \in \mathbb{R}^{2\times K}$
$\mathcal{O}_{\text{AO}} \leftarrow \emptyset$
$\mathcal{O}_{\text{ACT}} \leftarrow \mathcal{O}_{\text{STAT}}$
$\mathcal{P} \leftarrow [\mathbf{p}_0]$
$k \leftarrow 0$
while $\|\mathbf{p}_k - \mathbf{g}\|_2 > R_{\text{GOAL}}$ **do**
    $\mathcal{C}_k \leftarrow \text{sampleCandidates}(\mathbf{p}_k)$
    evaluate $j(\mathbf{c})$ for all $\mathbf{c} \in \mathcal{C}_k$
    $\mathbf{c}_k^* \leftarrow \arg\min_{\mathbf{c}\in\mathcal{C}_k} j(\mathbf{c})$
    if $j(\mathbf{c}_k^*) \geq j(\mathbf{p}_k)$ **then** ▷ local minimum detected
        add $\text{AO}(\mathbf{p}_k)$ to $\mathcal{O}_{\text{AO}}$
        $\mathcal{O}_{\text{ACT}} \leftarrow \mathcal{O}_{\text{STAT}} \cup \mathcal{O}_{\text{AO}}$
        $\mathcal{P} \leftarrow [\mathbf{p}_0]$ ▷ full trajectory restart
        $k \leftarrow 0$
    else
        $\mathbf{p}_{k+1} \leftarrow \mathbf{c}_k^*$
        append $\mathbf{p}_{k+1}$ to $\mathcal{P}$
        $k \leftarrow k + 1$
    end if
end while
return $\mathcal{P}$

Here, `sampleCandidates`$(\mathbf{p}_k)$ denotes the generation of the candidate set $\mathcal{C}_k$ around the current position $\mathbf{p}_k$ using the sampling radius R_B and the fixed set of candidate directions defined by the RAPF planner.

2.3 Rover Motion Model and Discrete Action Constraints

The RAPF formulation in the previous section describes navigation as a sequence of local candidate selections in the plane. This representation is useful for explaining the planning logic, since the planner evaluates candidate positions, navigation costs, and obstacle clearance. However, it should not be interpreted as a complete model of the rover's low-level motion capabilities.

The Lunar Zebro rover is not a holonomic platform. Its motion is constrained by its current heading, and it cannot command arbitrary lateral displacement in the plane. In practice, the rover can only execute a limited set of elementary actions. These actions are represented by the finite set

$$\mathcal{A} = \{\text{forward}, \text{backward}, \text{turn\_left}, \text{turn\_right}, \text{arc\_left}, \text{arc\_right}\}.$$

Each action $a \in \mathcal{A}$ corresponds to a short executable manoeuvre. The forward and backward actions translate the rover approximately along or opposite to its current heading. The turn_left and turn_right actions mainly change the rover heading, while arc_left and arc_right combine translation with a heading change. This finite action set captures the non-holonomic nature of the rover: the rover can move through short heading-constrained manoeuvres, but it cannot instantaneously command an arbitrary planar displacement.

This distinction is important because the planning representation and the executable motion representation do not operate at the same level of abstraction. The RAPF planner

selects candidate positions that define where the rover should move next in order to reduce the navigation cost. The physical rover, however, can only realize such motion through the elementary action set $\mathcal{A}$. Therefore, a planned displacement should be understood as a planning-level transition, not as an arbitrary velocity or displacement command that can be executed directly.

In the later swarm navigation architecture, this gap between desired motion and executable rover motion is handled explicitly by the motion primitive execution layer. In that layer, the finite primitive set $\mathcal{U}$ is introduced. A motion primitive may consist of one elementary action or of a short sequence of actions $a \in \mathcal{A}$. For example, a primitive may represent a single forward step, a backward recovery step, an arc action, or a short sequence that combines turning and translation. Thus, $\mathcal{A}$ represents the rover's elementary action repertoire, while $\mathcal{U}$ represents the higher-level set of candidate executable manoeuvres used during primitive selection.

The early single-agent chapters focus on the planning behaviour of RAPF and I-RAPF: goal reachability, obstacle avoidance, local-minima recovery, and replanning under sensing. The detailed conversion from planned or desired motion to executable non-holonomic manoeuvres is therefore deferred to the swarm execution layer, where it becomes necessary to compare continuous desired velocities against finite executable motion primitives.

2.4 Modeling the Lunar Navigation Landscape

The effectiveness of the Robust Artificial Potential Field (RAPF) logic is fundamentally tied to the geometry and density of the navigation landscape. To transition from abstract potential fields to mission-realistic performance, the environment must be modeled according to the physical constraints and geological characteristics of the lunar surface. This section establishes the geologically-grounded framework used to define the obstacles that generate the repulsive potentials (j_o) described in Section 2.2.2.

2.4.1 Geological Hazard Distributions

Obstacle dimensions are determined from empirical size-frequency distributions of lunar surface features obtained from Apollo, Chang'e, and Luna missions. It has been established that the cumulative fractional area covered by rocks larger than a given diameter D follows an exponential distribution:

$$F_R(D) = K_R e^{-Q_R(K_R)\cdot D} \tag{2.15}$$

where K_R denotes the rock abundance (percentage of surface covered by rocks), and Q_R is an empirically fitted coefficient. This statistical model allows for the derivation of the cumulative number of rocks per square meter, providing a physically grounded basis for the obstacle sets $\mathcal{O}_{\text{act}}$ used in the navigation cost functions.

2.4.2 Traversability Thresholds and Potential Sources

For the Lunar Zebro microrover, geological features are classified based on mechanical clearance limits. Rocks with diameters greater than $D_{crit} = 6.5$ cm (corresponding to a height of approximately 35 mm) are treated as hazards. In the context of the RAPF formulation, these features act as sources of infinite potential within the hard safety boundary R_l, while smaller features are considered traversable noise. Craters are modeled using the same statistical approach to ensure consistency across feature types.

2.4.3 Complexity Grading and Environmental Benchmarks

To systematically evaluate how the RAPF logic and swarm coordination strategies scale with environmental complexity, five benchmark scenarios were established. A critical experimental control in these scenarios is that the **total obstacle coverage is held constant** (rocks: ≈1.8%, craters: ≈11%). By fixing overall coverage while varying the number of obstacles, the simulations isolate the effects of obstacle density and size distribution on path planning performance.

The scenarios are divided into foundational optimization sets and interstitial generalization sets to ensure the robustness of the coordination logic across a continuous spectrum of lunar clutter (Table 2.1).

Table 2.1: Obstacle distributions for the five simulation scenarios. Scenarios 1, 3, and 5 represent the foundational baseline, while 2 and 4 are utilized for generalization testing.

Scenario	Original Label	Rocks	Craters	Total	Purpose
1	A	42	38	80	Optimization
2	–	65	35	100	Generalization
3	B	88	32	120	Optimization
4	–	112	28	140	Generalization
5	C	137	24	161	Optimization

2.4.4 Spatial Configuration and Map Parameters

To ensure reproducibility across all 500 simulation trials, a standardized spatial configuration was adopted. The global workspace is defined as a 30×30 m^2 area, with geological hazards distributed within a 20×20 m^2 central region. The rover begins each mission at a fixed starting position $(x_s, y_s) = (2, 2)$, navigating toward a target goal centered at $(x_g, y_g) = (28, 28)$ with a radius of 0.5 m. While the specific coordinates of rocks and craters are randomly generated for each trial to test the planner's adaptability, the underlying obstacle abundance remains strictly controlled per the densities defined in Table 2.1.

2.5 Chapter Summary

This chapter established the foundations needed to evaluate and extend potential-field-based navigation for lunar micro-rovers. The key takeaways are summarized below:

- **APF evolution**: Classical APF provides a lightweight and reactive navigation foundation, but is prone to local minima, oscillations, and deadlock-like behaviour in cluttered environments.
- **RAPF as baseline**: RAPF improves classical APF by replacing direct gradient following with cost-based local sampling and by introducing artificial obstacles as an explicit trap-escape mechanism.
- **Discrete candidate selection**: Rather than commanding arbitrary continuous motion, the planner evaluates a finite candidate set around the rover. This transforms navigation into a structured local search process.

- **Rover motion constraints**: The physical rover cannot execute arbitrary planar displacements. Its motion is constrained by heading and by a finite set of elementary actions, motivating the later motion primitive execution layer.
- **Lunar-inspired benchmark model**: The simulated navigation environments are based on empirical rock and crater distributions, with scenario difficulty varied through obstacle density while keeping total obstacle coverage controlled.

The following chapter builds on these foundations by developing the improved single-agent planner used in the remainder of the thesis. It first optimizes the RAPF parameterization and then introduces algorithmic improvements for physical path validity and more efficient local-minimum recovery, resulting in Improved RAPF (I-RAPF).

Chapter 3

RAPF Optimization and Algorithmic Improvements

The previous chapter introduced the baseline Robust Artificial Potential Field (RAPF) planner and the lunar-inspired benchmark environment used throughout this thesis. While this baseline formulation demonstrated the potential of combining artificial potential fields with bacterial sampling and artificial-obstacle-based local-minima escape, it did not yet constitute the final planner used in the remainder of this thesis. Additional work was required to improve its robustness, computational efficiency, and physical consistency.

This chapter focuses on the construction of that improved planner. The goal is not yet to evaluate RAPF in its final intended deployment setting, but to develop and refine the planning algorithm itself in a controlled environment. For this reason, the experiments in this chapter are performed under full map knowledge. This allows the effects of parameter choices and algorithmic modifications to be isolated from additional complexities introduced by online sensing, partial observability, and repeated replanning during execution.

The improvement process consists of two complementary parts. First, the parameterization of RAPF is systematically optimized using Bayesian hyperparameter search. Second, algorithmic refinements are introduced to address limitations that became apparent during tuning and controlled benchmarking. In particular, attention is given to improving collision validation, reducing unnecessary replanning effort, and increasing robustness in cluttered environments.

A subset of scenarios, namely A–C, is used in this chapter for optimization and controlled comparison. These scenarios were selected to provide a representative set of environment structures while keeping repeated hyperparameter evaluations computationally manageable. The broader scenario set is reserved for the next chapter, where the resulting planner is evaluated in a more realistic active-sensing navigation loop. This separation is intentional: the present chapter is concerned with algorithm development, whereas the next chapter addresses system-level evaluation under practical operating conditions.

The outcome of this chapter is a finalized improved RAPF formulation, referred to hereafter as *Improved RAPF* (I-RAPF). This name is used to distinguish the refined planner developed in this work from the baseline RAPF formulation introduced earlier and to provide a clear reference for the comparative results presented in the remainder of the thesis.

The chapter is structured as follows. Section 3.1 summarizes the main limitations of the baseline RAPF planner that motivated the present work. Section 3.2 introduces the optimization framework, including the hyperparameter search space and the evolution of

the objective function. Section 3.3 presents the main algorithmic improvements introduced on top of the baseline formulation. Section 3.4 evaluates these improvements in the controlled full-map setting and compares the resulting planner variants. Finally, Section 3.5 discusses the main findings, and Section 3.6 summarizes the chapter and connects it to the sensing-based evaluation.

3.1 Limitations of the Baseline RAPF

The baseline RAPF algorithm, as introduced in the previous chapter, combines potential-field guidance with bacterial sampling and virtual-obstacle insertion to escape local minima. Although this formulation provides a robust starting point, three limitations became clear during early experimentation.

First, the planner performance is highly sensitive to the choice of hyperparameters. The interaction between attractive and repulsive field terms, the candidate sampling behaviour, and the local-minima recovery mechanism creates a complex multi-dimensional design space in which manual tuning is difficult and sub-optimal configurations are easily obtained.

Second, visual inspection of early planning results revealed a geometric inconsistency in the candidate validation procedure. Because collision checking initially considered only discrete candidate endpoints, the planner could accept steps whose straight-line connection implicitly passed through thin obstacles or crater rims. This tunneling effect produced trajectories that appeared valid in the discrete representation but were not physically feasible.

Third, the original stagnation recovery behaviour relied on full-trajectory replanning. Whenever the planner became trapped and inserted a new virtual obstacle, the entire previously computed trajectory was discarded and planning restarted from the beginning. In cluttered environments this behaviour was computationally inefficient, as large valid prefixes of the trajectory were recomputed even when only a local suffix had become invalid.

These limitations motivated the combined optimization and algorithmic improvement strategy presented in the remainder of this chapter.

3.2 Optimization Framework

The tuning of RAPF is formulated as a black-box optimization problem over the planner hyperparameters. Let $\boldsymbol{\theta}$ denote the vector of tunable RAPF parameters. The objective is to find a configuration $\boldsymbol{\theta}^*$ that minimizes an aggregate trial score $j_{\text{trial}}(\boldsymbol{\theta})$ obtained from repeated simulated navigation episodes. This optimization problem is non-convex, derivative-free, and computationally expensive, since each objective evaluation requires a batch of full planning-and-execution rollouts. The optimization framework therefore consists of three parts: a bounded hyperparameter search space, an evaluation objective defined over batches of environments, and a sample-efficient optimization algorithm.

3.2.1 Parameter Search Space

The RAPF planner contains several tunable parameters that influence the shape of the potential field and the behaviour of the sampling-based trajectory construction. Existing RAPF and BAPF-based approaches define the functional role of these parameters, such as controlling the magnitude and spatial decay of attractive and repulsive potentials, but

do not prescribe explicit parameter ranges. Instead, parameter values are typically tuned empirically for the specific environment and robot configuration, as also observed in [9].

In this work, bounded parameter ranges are proposed based on three guiding principles. First, *numerical stability and gradient conditioning*: ranges are selected such that the resulting potential gradients remain well-scaled across the workspace, avoiding vanishing attraction or excessively steep repulsion. Second, *physical and geometric consistency*: parameters are bounded relative to the robot size and environment scale to ensure meaningful clearance and feasible motion. Third, *the trade-off between exploration and computational cost*: sampling-related parameters are chosen to balance directional exploration capability against the cost of evaluating additional candidates.

Table 3.1: Hyperparameter search space.

Parameter	**Type**	**Range**	**Purpose**
A_a	Float	[2.0, 20.0]	Goal attraction strength
M_a	Float	$[2 \cdot 10^{-4}, 8 \cdot 10^{-4}]$	Goal decay rate
A_o	Float	[2.0, 20.0]	Obstacle repulsion strength
M_o	Float	[50.0, 100.0]	Obstacle decay rate
R_u	Float	[1.0, 2.5]	Obstacle influence radius
R_B	Float	[0.25, 0.6]	Sampling radius / step size
N_B	Int	[4, 16]	Number of samples per planning step
R_{AO}	Float	[0.2, 0.8]	Artificial-obstacle radius

The attraction and repulsion parameters (A_a, M_a) and (A_o, M_o) govern the magnitude and spatial decay of the respective potential fields in Equations 2.11 and 2.12. The decay parameters operate on different spatial scales: the attractive potential is evaluated over the entire workspace, whereas the repulsive potential is only active in the vicinity of obstacles. As a result, M_a must be significantly smaller than M_o to produce comparable gradient magnitudes.

The obstacle influence radius R_u determines the spatial extent of repulsive forces and therefore directly affects the trade-off between safety and path efficiency. Larger values increase clearance but may lead to overly conservative trajectories.

The sampling parameters R_B and N_B define the local exploration strategy of the planner. The sampling radius R_B sets the step size, while N_B controls the angular resolution of the candidate directions. Increasing these values improves exploration capability but also increases computational cost.

Finally, the artificial-obstacle radius R_{AO} controls the strength of the local-minima recovery mechanism by determining how strongly previously explored regions are penalized during replanning.

3.2.2 Evolution of the Objective Function

For a parameter configuration $\boldsymbol{\theta}$, the optimization objective is defined over a batch of simulated environments rather than a single rollout. In the main tuning study, each Optuna trial consisted of 20 episodes for each of scenarios A, B, and C, resulting in 60 episode evaluations per trial. The aggregate trial score was computed as

$$j_{\text{trial}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} s_i, \tag{3.1}$$

where s_i denotes the scenario score obtained in environment i and N is the constant number of environments in the evaluation batch.

It is important to note that j_{trial} is not intended as a physically derived cost function. The quantities used to evaluate planner behaviour, such as path length, planning effort, restart events, artificial-obstacle insertions, and collisions, have different physical meanings and units. For example, l_{path} is measured in meters, while e, n_{restart}, n_{ao}, and $n_{\text{collision}}$ are event counts. These quantities are therefore not summed as raw physical terms. Instead, each component is multiplied by a fixed coefficient and interpreted as a weighted, dimensionless penalty contribution to a scalar performance score. The objective is thus an engineered evaluation metric used to rank parameter configurations during optimization.

The tiered structure of the score further imposes a strict preference ordering between qualitatively different outcomes. Planner failures receive the largest penalty, execution failures receive an intermediate penalty, and successful navigation episodes are ranked according to their weighted score. In this way, the optimization prioritizes feasibility and goal attainment first, and only then distinguishes between successful runs on the basis of efficiency and robustness.

For each environment, the scenario score was computed according to the following three-tier rule.

- **Planner failure:** if the planner failed to generate a valid path, a large fixed penalty was assigned,
$$s_i = 20000. \tag{3.2}$$
- **Execution failure:** if a path was generated but the agent did not reach the goal during simulation, a distance-based penalty was assigned,
$$s_i = 5000 + 100\, d_{\text{goal}}, \tag{3.3}$$
where d_{goal} denotes the final distance between the agent and the goal at the end of the simulation.
- **Successful navigation:** if the agent reached the goal, the scenario score was computed as
$$s_i^{\text{succ}} = l_{\text{path}} + c_{\text{thinking}} + c_{\text{repair}} + c_{\text{collision}}. \tag{3.4}$$

Here, l_{path} denotes the executed path length, c_{thinking} the weighted penalty associated with planning effort, c_{repair} the weighted penalty associated with local-minima recovery actions, and $c_{\text{collision}}$ the weighted penalty associated with collisions. These terms are written separately to make the structure of the score explicit: each represents a dimensionless contribution to the final scenario score after application of the relevant weighting coefficients.

While the planner-failure and execution-failure tiers remained unchanged, the formulation of the *successful-navigation tier* evolved during the optimization study as the relative importance of computational effort and recovery actions became better understood.

Initial successful-navigation tier During the initial optimization phase, the success-tier score emphasized path quality while strongly penalizing recovery complexity:

$$s_{i,\text{I}}^{\text{succ}} = l_{\text{path}} + 0.001\, e + 50\, n_{\text{restart}} + 100\, n_{\text{ao}} + 2000\, n_{\text{collision}}. \tag{3.5}$$

In this expression, e denotes the number of internal planning iterations, n_{restart} the number of restart events during recovery, n_{ao} the number of artificial obstacles created,

and $n_{\text{collision}}$ the number of collisions. The numerical coefficients do not imply dimensional equivalence between these quantities. Rather, they define their relative contribution to the scalar score and were selected heuristically to reflect the design priorities of the planner. In this first phase, restart events and artificial-obstacle placement were treated as strongly undesirable, thereby favouring conservative behaviour that attempted to avoid repair actions altogether.

Refined successful-navigation tier In the next refinement phase, the success-tier score was adjusted to penalize the total number of potential evaluations more directly:

$$s_{i,\text{II}}^{\text{succ}} = l_{\text{path}} + 0.001\,(e \cdot N_B) + 10\,n_{\text{restart}} + 2000\,n_{\text{collision}}. \tag{3.6}$$

Here, the effort term was scaled by N_B, the constant number of candidate points evaluated per planning iteration, yielding a better approximation of the true computational burden of the planner. At the same time, artificial-obstacle placement was no longer penalized explicitly, reflecting the insight that artificial obstacles are not failures in themselves, but valid tools for escaping local minima. The weighting on restart events was also reduced, indicating a shift away from discouraging recovery altogether and towards rewarding efficient recovery.

Final successful-navigation tier In the final optimization phase, the same structure was retained, but the penalty on restart events was reduced further:

$$s_{i,\text{III}}^{\text{succ}} = l_{\text{path}} + 0.001\,(e \cdot N_B) + 1\,n_{\text{restart}} + 2000\,n_{\text{collision}}. \tag{3.7}$$

This final formulation reflects a *fail-fast* philosophy. Recovery events were no longer interpreted as undesirable outcomes on their own. Instead, the optimization rewarded planners that resolved local minima efficiently, provided that the resulting path length and total computational effort remained low. The score therefore evolved from a formulation that discouraged recovery actions toward one that accepted them as useful mechanisms, as long as they contributed to successful and efficient navigation.

3.2.3 Tree-Structured Parzen Estimator (TPE)

Given the bounded search space and the expensive black-box objective defined above, a sample-efficient optimization method is required. The hyperparameter optimization was therefore performed using the Tree-Structured Parzen Estimator (TPE) algorithm as implemented in the Optuna framework. TPE is a Bayesian optimization method that models the relationship between hyperparameter configurations and their resulting objective values based on previous trials.

Instead of sampling parameters uniformly at random, TPE builds two probabilistic density models: one representing configurations that yielded good performance and another representing less promising configurations. New candidate parameters are then sampled from regions that maximize the ratio between these densities, biasing the search towards areas of the parameter space that are more likely to improve the objective.

This choice is well suited to the present tuning problem for three reasons. First, the trial objective is expensive to evaluate because each trial requires multiple full episode rollouts. Second, the objective is derivative-free and highly non-linear, making gradient-based methods inapplicable. Third, the number of tunable parameters is moderate but their interactions are non-trivial, which makes purely exhaustive or uniform random search inefficient. TPE therefore provides a practical compromise between exploration of the search space and concentration of evaluations in promising regions.

3.3 Algorithmic Improvements to RAPF

The optimization framework in Section 3.2 improved the parameterization of RAPF, but it also exposed limitations that could not be solved by parameter tuning alone. In particular, two structural issues remained: first, the discrete candidate validation could accept geometrically infeasible segments, and second, the original recovery mechanism discarded more of the trajectory than necessary after a local-minimum event. This section therefore focuses on the algorithmic changes introduced on top of the optimized baseline.

The purpose of these changes is not to create several separate planner variants for later use, but to define the final planner used in the rest of this thesis. After midpoint collision checking and partial-trajectory repair are introduced, the resulting method is referred to as *Improved RAPF* (I-RAPF). The experimental evaluation in Section 3.4 therefore compares the baseline RAPF formulation against I-RAPF, rather than treating each internal design decision as a separate planner in the main results.

3.3.1 Midpoint Collision Checking

Visual analysis of the early optimization results revealed a critical physical inconsistency known as *tunneling*. Because the bacterial sampling radius R_B defines a discrete step between candidate waypoints, the planner could occasionally accept a candidate whose endpoint was collision-free even though the straight-line segment from the current position to that candidate crossed a thin obstacle or crater rim. Such a trajectory is valid in the discrete waypoint representation, but not in the continuous geometry of the environment.

To address this issue, a midpoint collision check was added to the candidate evaluation step. For every candidate bacterium $\mathbf{c} \in \mathcal{C}_k \subset \mathbb{R}^{2\times 1}$ generated from the current planner position $\mathbf{p}_k \in \mathbb{R}^{2\times 1}$, the midpoint $\mathbf{m}(\mathbf{c}) \in \mathbb{R}^{2\times 1}$ is computed as

$$\mathbf{m}(\mathbf{c}) = \frac{\mathbf{p}_k + \mathbf{c}}{2}. \tag{3.8}$$

The candidate is rejected, equivalently assigned $j(\mathbf{c}) = \infty$, if this midpoint lies inside the hard physical radius of any obstacle. The check is therefore a lightweight approximation of segment validity: it prevents the planner from accepting candidate steps that would pass through an obstacle even when both endpoints are individually collision-free.

Figure 3.1 illustrates this failure mode. The waypoints A and B are both outside the obstacle boundary, but the connecting segment intersects the obstacle region. Without an additional segment-level check, the planner incorrectly accepts the motion. The midpoint check removes this source of physically infeasible success without requiring a smaller sampling radius. This is important because reducing R_B would also increase the number of planning iterations required to cover the same distance.

3.3.2 Partial-Trajectory Repair through Influence-Horizon Pruning

The second structural improvement concerns the response to local minima. In the baseline RAPF formulation, the planner escapes stagnation by inserting an artificial obstacle at the stagnation point and then restarting the full trajectory computation. This full-restart mechanism is robust, but inefficient: when a local minimum occurs near the end of a path, the planner discards the entire valid prefix even though the newly inserted artificial obstacle only modifies the potential field locally.

I-RAPF replaces this full restart by partial-trajectory repair. The key idea is that the influence of a newly inserted artificial obstacle is spatially bounded. For an artificial

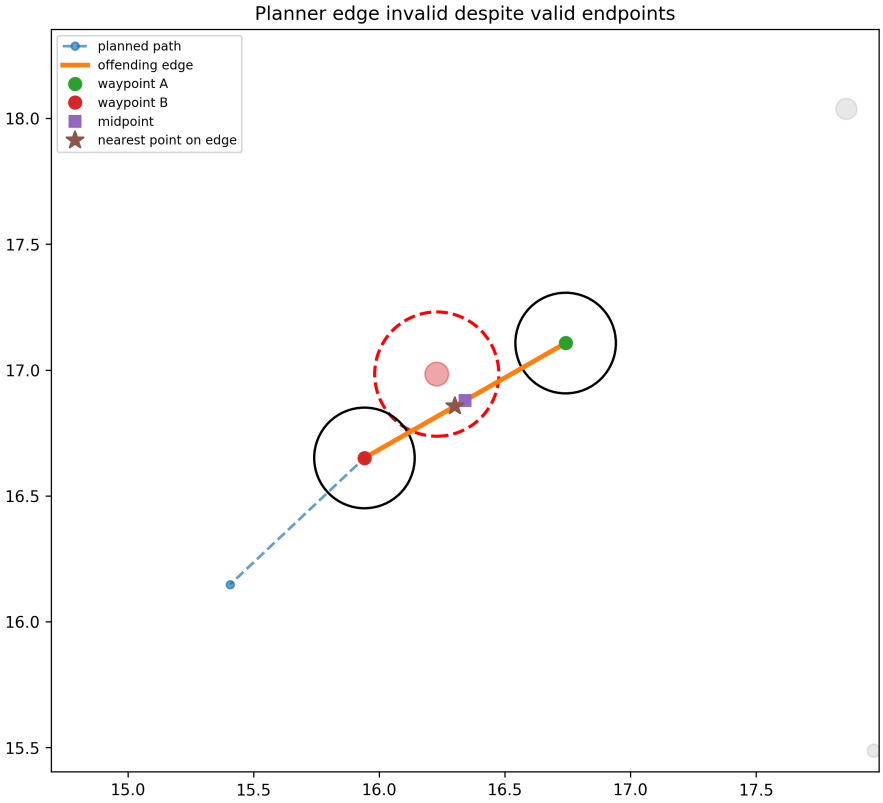

Figure 3.1: Tunneling example where two collision-free waypoints produce an invalid edge that intersects an obstacle. The midpoint collision check rejects this candidate segment.

obstacle with radius R_{AO} placed at the stagnation position $\mathbf{p}_{\text{stuck}} \in \mathbb{R}^{2\times 1}$, the affected region is defined by the influence horizon

$$H_{\text{influence}} = R_{AO} + R_u, \tag{3.9}$$

where R_u is the obstacle influence radius used in the repulsive potential. Waypoints outside this horizon are not directly affected by the new artificial obstacle and can therefore be retained.

Stagnation is defined by the absence of an improving candidate, rather than by zero velocity. Let $\mathbf{c}_k^* \in \mathbb{R}^{2\times 1}$ denote the lowest-cost candidate at planning iteration k. A local-minimum event is triggered when

$$j(\mathbf{c}_k^*) \geq j(\mathbf{p}_k). \tag{3.10}$$

After the artificial obstacle is inserted, the planner searches the existing path backwards and selects the last waypoint outside the influence horizon as a repair anchor. The affected suffix is removed, while the valid prefix is preserved. Planning then resumes from the retained anchor point using the updated obstacle set.

Algorithm 2 summarizes the resulting I-RAPF procedure. Compared with the baseline RAPF planner, the method includes both midpoint collision checking during candidate evaluation and influence-horizon pruning during local-minimum recovery.

Algorithm 2 Improved RAPF (I-RAPF)

Input: start position $\mathbf{p}_0 \in \mathbb{R}^{2\times 1}$, goal position $\mathbf{g} \in \mathbb{R}^{2\times 1}$, static obstacle set $\mathcal{O}_{\text{STAT}}$
Output: path $\mathcal{P} \in \mathbb{R}^{2\times K}$
$\mathcal{O}_{\text{AO}} \leftarrow \emptyset$
$\mathcal{O}_{\text{ACT}} \leftarrow \mathcal{O}_{\text{STAT}}$
$\mathcal{P} \leftarrow [\mathbf{p}_0]$
$k \leftarrow 0$
while $\|\mathbf{p}_k - \mathbf{g}\|_2 > R_{\text{GOAL}}$ **do**
    $\mathcal{C}_k \leftarrow \text{sampleCandidates}(\mathbf{p}_k)$
    for all $\mathbf{c} \in \mathcal{C}_k$ **do**
        if $\text{midpointValid}(\mathbf{p}_k, \mathbf{c}, \mathcal{O}_{\text{ACT}}) = \text{false}$ **then**
            $j(\mathbf{c}) \leftarrow \infty$
        else
            evaluate $j(\mathbf{c})$
        end if
    end for
    $\mathbf{c}_k^* \leftarrow \arg\min_{\mathbf{c}\in\mathcal{C}_k} j(\mathbf{c})$
    if $j(\mathbf{c}_k^*) \geq j(\mathbf{p}_k)$ **then** ▷ local minimum detected
        $\mathbf{p}_{\text{stuck}} \leftarrow \mathbf{p}_k$
        add $\text{AO}(\mathbf{p}_{\text{stuck}})$ to $\mathcal{O}_{\text{AO}}$
        $\mathcal{O}_{\text{ACT}} \leftarrow \mathcal{O}_{\text{STAT}} \cup \mathcal{O}_{\text{AO}}$
        $H_{\text{influence}} \leftarrow R_{\text{AO}} + R_u$
        $a \leftarrow \text{repairAnchor}(\mathcal{P}, \mathbf{p}_{\text{stuck}}, H_{\text{influence}})$
        $\mathcal{P} \leftarrow [\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_a]$ ▷ retain valid prefix
        $k \leftarrow a$
    else
        $\mathbf{p}_{k+1} \leftarrow \mathbf{c}_k^*$
        append $\mathbf{p}_{k+1}$ to $\mathcal{P}$
        $k \leftarrow k + 1$
    end if
end while
return $\mathcal{P}$

This repair mechanism changes the computational character of RAPF. Instead of resolving the entire path after every local-minimum event, I-RAPF only recomputes the suffix that may be affected by the new artificial obstacle. The planner therefore preserves useful trajectory structure while still modifying the local potential field enough to escape stagnation.

3.4 Experimental Evaluation

The experimental evaluation has two goals. First, it verifies that the hyperparameter optimization process identifies a stable operating regime for RAPF. Second, it evaluates the complete improved planner, I-RAPF, against the baseline RAPF formulation. The comparison is therefore made at the planner level: RAPF denotes the optimized baseline formulation with full-trajectory restart, while I-RAPF denotes the final method introduced in Section 3.3, including midpoint collision checking and partial-trajectory repair.

The experiments in this chapter use scenarios A–C. These scenarios are treated as development scenarios and are used for hyperparameter optimization and controlled com-

parison of algorithmic variants. The selection was made to ensure a representative range of obstacle configurations while keeping the computational cost of repeated evaluations tractable.

Scenarios D–E are not included in this stage, as they are reserved for the evaluation presented in Chapter 4. This separation allows the final planner, I-RAPF, to be assessed on previously unused environments, providing a clearer indication of its generalization performance under realistic sensing conditions.

3.4.1 Optimization Behaviour

The optimization study was first used to identify a suitable operating regime for the RAPF parameters. Figure 3.2 shows the optimization history of the broad Phase I study. The objective value decreases substantially during the first part of the search and then stabilizes around a narrower region of the parameter space. This suggests that the planner admits a consistent high-performing parameter regime, rather than depending on a single fragile parameter combination.

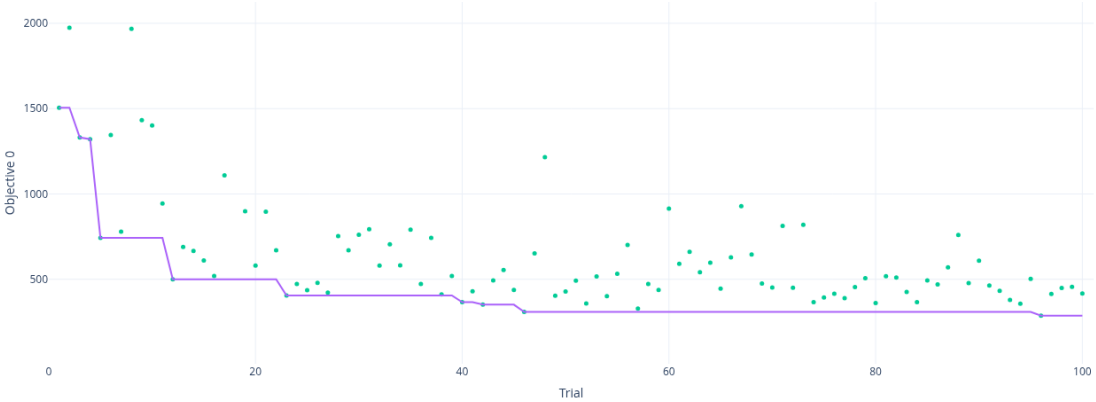

Figure 3.2: Optimization history of the Phase I study. The TPE sampler reduces the objective over 100 trials and identifies a stable high-performing parameter regime.

After the broad search, the search space was narrowed and a final optimization cycle was performed on the improved formulation. Figure 3.3 reports the resulting hyperparameter importance. The most important parameters are those that directly control the balance between attraction, repulsion, and local exploration.

Table 3.2 summarizes the key tuned parameters retained for the final planner. The values indicate a moderate attractive pull, strong obstacle repulsion, and sufficient candidate density for local exploration.

Table 3.2: Key optimized parameters retained for I-RAPF.

Parameter	**Value**	**Observation**
A_a	9.13	Moderate goal pull prevents overshoot.
A_o	18.70	Strong repulsion encourages safe clearance.
N_B	11	Sufficient candidate density for local exploration.

The important outcome of this optimization stage is not only the final numerical parameter set, but also the observation that similar parameter regimes remained competitive after stricter geometric validation and the revised objective formulation were introduced.

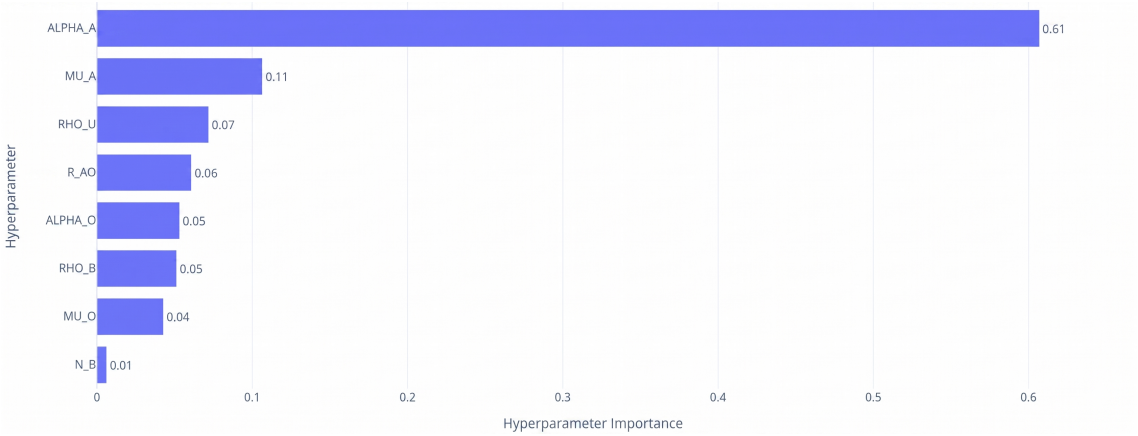

Figure 3.3: Hyperparameter importance derived from the final optimization cycle.

This supports the use of the optimized parameter set as the basis for the final I-RAPF planner.

3.4.2 RAPF versus I-RAPF

This subsection summarizes the effect of the parameter optimization and algorithmic improvements introduced in this chapter. The comparison is not intended as the final benchmark of the planner, but as a controlled development evaluation on Scenarios A–C. These scenarios are used to compare the reference RAPF success rates reported by Manteaux et al. [9], the RAPF planner with the tuned parameter configuration, and the final improved planner, I-RAPF.

Table 3.3: Success rate comparison between reference RAPF, tuned RAPF, and I-RAPF on development scenarios (A–C).

Scenario	**Reference RAPF (%)**	**Tuned RAPF (%)**	**I-RAPF (%)**
Scenario A	96.4	98.2	**100.0**
Scenario B	93.8	98.6	**100.0**
Scenario C	91.8	98.4	**99.8**

Table 3.3 shows a consistent increase in success rate from the reference RAPF values to the tuned RAPF configuration, and from the tuned configuration to I-RAPF. The tuned RAPF planner already improves over the reference values, indicating that the selected parameter configuration provides a stronger baseline for the scenario family considered in this chapter. The additional improvement obtained by I-RAPF is smaller in absolute percentage points, but is important because it reflects algorithmic changes rather than parameter tuning alone.

During development, the midpoint collision check was introduced to prevent accepted trajectory segments from passing through obstacles between two successive sampled positions. After this modification, no obstacle penetrations caused by this step-size effect were observed in the tested development scenarios. This observation is treated as qualitative development evidence; the systematic evaluation of collision-related metrics is deferred to the benchmark chapter.

Planning effort is measured as the number of planner iterations. Each iteration corresponds to the evaluation of a candidate trajectory segment within the RAPF sampling

process. Since both planners use the same implementation structure, this metric provides a consistent proxy for computational effort.

Table 3.4: Planning effort comparison between tuned RAPF and I-RAPF on development scenarios (A–C).

Scenario	Tuned RAPF Iterations	I-RAPF Iterations	Reduction
Scenario A	291.7	**73.3**	**74.9%**
Scenario B	311.8	**74.7**	**76.0%**
Scenario C	336.3	**76.8**	**77.2%**

The clearest quantitative improvement is observed in planning effort. As shown in Table 3.4, I-RAPF reduces the average number of internal planning iterations by approximately 75–77% compared with the tuned RAPF baseline. This reduction follows from the partial-repair mechanism: valid trajectory prefixes are preserved instead of being recomputed after every recovery event. The effect becomes slightly stronger as obstacle density increases, because cluttered environments produce more recovery events and therefore provide more opportunities for repair to reduce computation.

Overall, the results support the use of I-RAPF as the final single-agent planner for the next stage of the thesis. I-RAPF preserves the reactive and sampling-based character of RAPF, but combines the optimized parameter set with trajectory repair, midpoint collision checking, and post-processing to improve reachability, physical validity, and computational efficiency.

3.5 Discussion

The results of this chapter show that improving RAPF required both parameter optimization and structural algorithmic changes. Hyperparameter tuning was necessary to identify a stable operating regime, but tuning alone could not resolve geometric inconsistencies or inefficient recovery behaviour. These limitations motivated the transition from optimized RAPF to I-RAPF.

The midpoint collision check addresses the gap between discrete waypoint planning and continuous robot motion. Without this check, a planner can appear successful while producing segments that intersect obstacles. This would inflate reported success rates and weaken the physical interpretation of the results. By rejecting candidates whose midpoint intersects an obstacle, I-RAPF provides a simple but effective safeguard against this failure mode.

Partial-trajectory repair addresses the computational cost of local-minimum recovery. The baseline full-restart mechanism is conservative, but it repeatedly recomputes valid path prefixes. I-RAPF instead uses the finite influence of a new artificial obstacle to determine which part of the path may be affected. This makes the recovery mechanism local rather than global, reducing planning effort while preserving useful trajectory structure.

The evaluation deliberately uses scenarios A–C as development scenarios. This means that the results in this chapter should be interpreted as controlled planner-development evidence rather than as the final generalization benchmark of the thesis. The final assessment on previously unused scenarios and under realistic partial observability is deferred to Chapter 4. This separation keeps the methodological role of the present chapter clear: it develops I-RAPF, while the next chapter evaluates it in the sensing-based navigation setting for which it is ultimately intended.

3.6 Chapter Summary

This chapter developed Improved RAPF (I-RAPF) from the baseline RAPF planner introduced in Chapter 2. The improvement process combined Bayesian hyperparameter optimization with two structural algorithmic refinements: midpoint collision checking and partial-trajectory repair. Midpoint collision checking improved the physical validity of generated paths by rejecting candidate transitions that could pass through obstacles between sampled waypoints. Partial-trajectory repair reduced unnecessary recomputation after local-minimum events by preserving valid trajectory prefixes and repairing only the affected suffix.

In controlled full-map experiments on the development scenarios, I-RAPF achieved near-perfect reachability and substantially reduced planning effort compared with the tuned restart-based RAPF formulation. These results establish I-RAPF as the finalized single-agent planner used in the remainder of the thesis.

The next chapter embeds I-RAPF in a sensing-based navigation loop. In that setting, the rover no longer has access to the complete obstacle map at the start of the episode. Instead, obstacles are discovered during execution, the active obstacle set is updated online, and the current route must be repeatedly validated, repaired, or replaced as new information becomes available.

Chapter 4

Sensing-Based Evaluation of I-RAPF

The previous chapter developed the final improved planner formulation used in this thesis, referred to as Improved RAPF (I-RAPF). That chapter considered the planner under full map knowledge in order to isolate the effects of hyperparameter optimization and algorithmic design choices in a controlled setting. While this was necessary for the systematic development of the planner, it does not yet reflect the conditions under which the rover is expected to operate in practice.

In realistic deployment, the environment is not known in advance. Obstacles are detected online through local sensing, and the planner must operate under partial and evolving map information. As a result, navigation is no longer a single full-map planning problem, but a reactive process in which planning, obstacle discovery, path validation, and replanning are repeatedly interleaved during execution.

This chapter therefore evaluates I-RAPF in a sensing-based navigation framework and compares it against Tuned RAPF as a controlled baseline. The objective is not to further redesign the planner itself, but to assess how the finalized I-RAPF planner behaves when embedded in a more realistic operational loop. In this setting, the rover incrementally constructs its map from sensed obstacles and triggers replanning whenever newly detected obstacles invalidate the current path or create previously unseen constraints on feasible motion.

In contrast to the optimization chapter, which used scenarios A–C for controlled tuning and algorithm comparison, the present chapter evaluates the planner variants on the full scenario set. The additional scenarios are included here to assess whether the behavior observed during development generalizes beyond the maps used during optimization and to reduce the risk of drawing conclusions from a limited set of environment configurations. This broader evaluation is therefore intended as the primary benchmark of I-RAPF's robustness under practical operating assumptions, with Tuned RAPF included as the reference baseline.

The sensing-based formulation also introduces phenomena that are absent in the full-map case, such as delayed obstacle discovery, local map inconsistencies, path interruption during execution, and repeated recovery from newly emerging blockages. These effects are essential to study, since they determine whether a planner that performs well in a controlled offline setting also remains reliable when deployed on a robot with limited perception.

The chapter is structured as follows. Section 4.1 introduces the sensing-based navigation loop and explains how the planner is embedded in a reactive perception–planning–execution framework. Section 4.2 presents the experimental setup and evaluation methodology for the full scenario set. Section 4.3 defines the evaluation metrics. Section 4.4 illustrates representative sensing-based navigation behaviour. Section 4.5 reports the bench-

mark results. Section 4.6 then discusses the interpretation of these results, and Section 4.7 summarizes the chapter and connects it to the swarm-navigation problem addressed next.

4.1 Navigation Loop

The sensing-based agent operates in a repeated perception–planning–advancement loop. At each simulation step, the agent first updates its local obstacle map using the currently available sensor information. The current planned path is then checked for validity with respect to the perceived obstacles. If the path is missing or no longer safe, the selected planner variant is invoked again from the agent's current position. Once a valid waypoint sequence is available, the agent advances toward the next waypoint and the cycle repeats.

This organization separates *planning* from *navigation management*. The planner computes a waypoint sequence from the current pose to the goal, while the agent controller manages sensing, path validation, backtracking, and progression along the path. The resulting system is reactive: new obstacle detections can invalidate the current path and trigger replanning during execution.

In the sensing-based setting, the rover does not have access to the complete static obstacle set $\mathcal{O}_{STAT}$. Instead, it plans using the active obstacle set $\mathcal{O}_{ACT}$, which contains the obstacles known to the rover at navigation step k. This set is updated whenever new obstacles enter the sensing range.

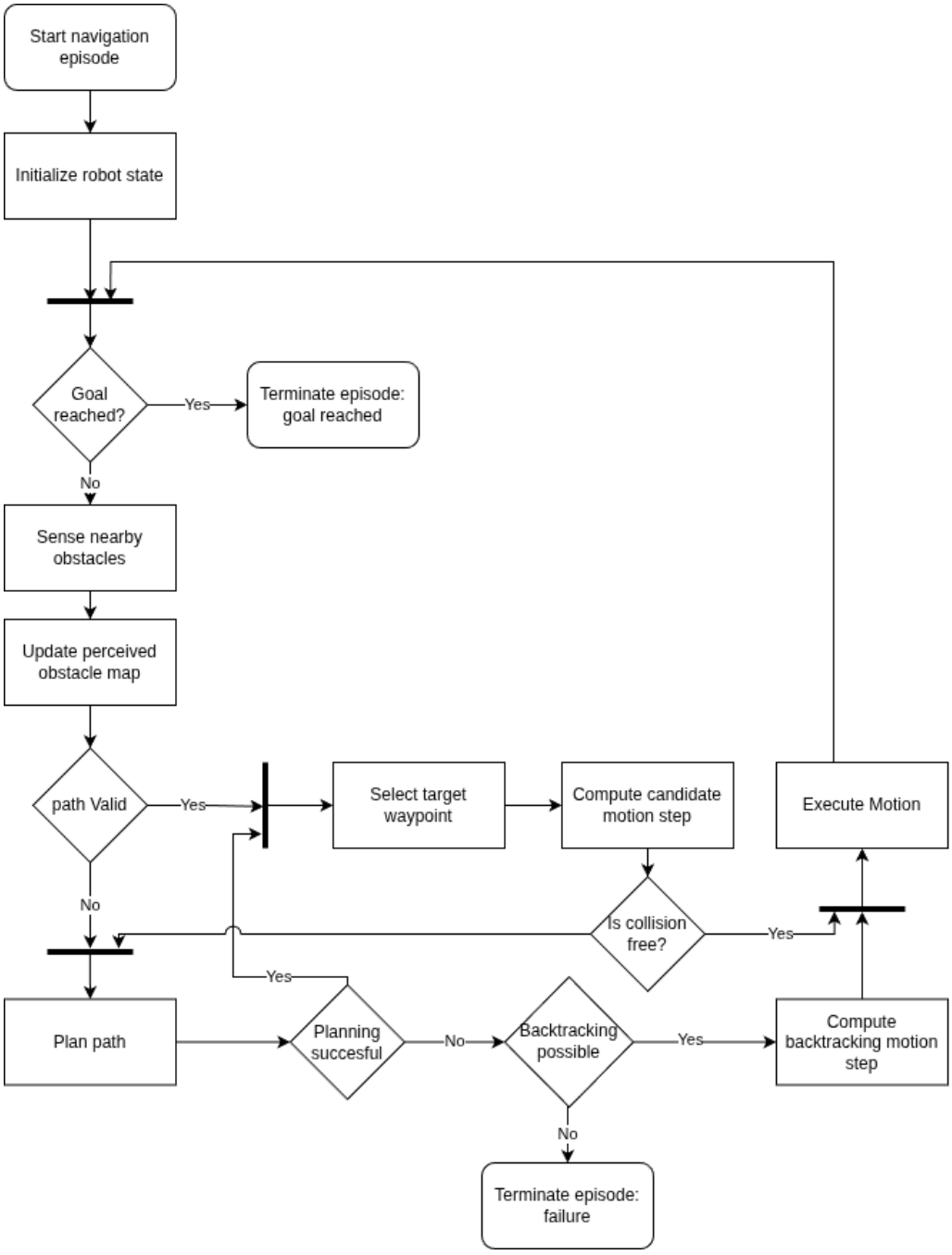

Figure 4.1: Flow diagram of the sensing-based RAPF navigation process. The agent incrementally updates its perceived obstacle map, validates the current path, replans when necessary, checks candidate motion steps before execution, and uses backtracking as a recovery mechanism when planning or local execution fails.

Figure 4.1 shows the complete control flow used during one navigation episode. If the current path remains valid after sensing, the agent selects a target waypoint, computes a candidate motion step, and executes that step only if it is collision-free. If the path is invalid, the planner is invoked using the updated active obstacle set. A successful planning attempt returns the agent to the path-following part of the loop. If planning fails, or if the candidate motion step is not collision-free, the agent attempts a backtracking step. The

episode terminates successfully when the goal is reached and terminates as a failure when neither replanning nor backtracking can produce a safe continuation.

4.1.1 Sensing and Map Construction

The agent perceives its environment through a local sensing mechanism, characterized by a finite sensing radius R_S. Within this region, obstacles are detected and incorporated into a local map representation.

Unlike the full-map assumption used in earlier chapters, the environment is revealed incrementally as the agent moves. This results in a progressively refined map, where previously unknown obstacles may appear along the planned trajectory.

The map is updated at each iteration of the navigation loop, ensuring that newly detected obstacles are immediately considered during path validation and replanning.

4.1.2 Path Validation and Replanning

Because obstacles are discovered during execution, a path that was safe when generated may become unsafe later in the episode. The navigation manager therefore checks whether the remaining planned path is still valid with respect to the active obstacle set $\mathcal{O}_{ACT}$.

When a newly detected obstacle lies sufficiently close to the remaining trajectory, the path is marked invalid. The selected planner variant is then invoked again from the agent's current position using the updated active obstacle set. If planning succeeds, the newly generated waypoint sequence replaces the previous path and execution continues. If planning fails, the navigation manager does not immediately terminate the episode, but first attempts an agent-side backtracking recovery step.

This validation-and-replanning mechanism allows the agent to operate under partial observability without committing to paths that were planned using outdated map information.

4.1.3 Tuned RAPF and I-RAPF as Planning Components

The planning component inside the navigation loop is based on the Robust Artificial Potential Field (RAPF) framework. This chapter compares two planner variants under the same sensing-based navigation conditions. The first variant is referred to as *Tuned RAPF*. It uses the baseline RAPF formulation from Manteaux et al. [9], but with the optimized parameter set obtained in Chapter 3. The second variant is *I-RAPF*, the finalized improved planner developed in Chapter 3, which uses the same tuned parameter set together with the algorithmic modifications introduced in that chapter.

This terminology is important for the interpretation of the results. The comparison in this chapter is not between the original parameterization of RAPF from prior work and the proposed planner. Instead, it compares a tuned RAPF baseline against I-RAPF. As a result, both variants are evaluated with the same parameter values, and the observed performance differences can be attributed primarily to the algorithmic improvements rather than to differences in parameter tuning.

The original RAPF work by Manteaux et al. does not provide a fully reproducible parameter configuration for the scenario family considered in this thesis, and reported values are tied to different environmental assumptions. Directly adopting parameters from prior work would therefore not provide a controlled baseline. Using the optimized parameter set for both variants gives a stronger comparison because the planner formulation, rather than the parameter choice, becomes the main experimental difference.

The formulation of RAPF itself is not reintroduced here. Instead, the focus is on how the two variants behave when embedded in the same sensing-based navigation loop. In both cases, the planner receives the current agent position, the goal position, and the active obstacle set $\mathcal{O}_{ACT}$, and returns either a waypoint sequence or a planning failure.

4.1.4 Artificial Obstacle Memory

To improve robustness under partial observability, the planner maintains a memory of previously generated artificial obstacles. These artificial obstacles are introduced during failed planning attempts, as described in Chapter 3, and are retained across replanning iterations in the sensing-based loop.

This persistence is useful because the agent may re-encounter the same problematic region while the perceived map is still incomplete. Without memory, repeated replanning attempts could return to the same local-minimum region. By retaining artificial obstacles across replans, the navigation loop carries forward information about previously encountered planning traps.

The mechanism is present in both planner variants. However, its effect can differ between Tuned RAPF and I-RAPF because the improved planner also contains the additional robustness mechanisms introduced in Chapter 3.

4.1.5 Step-wise Execution Strategy

Trajectory execution is performed in a step-wise manner. Rather than committing to the full planned trajectory, the agent selects a target waypoint on the current path and computes a candidate motion step toward that waypoint. This candidate step is checked against the active obstacle set before it is executed.

If the candidate motion step is collision-free, the robot state is updated and the navigation loop repeats. This gives the agent a receding-horizon execution structure: only a limited portion of the path is followed before sensing, path validation, and possible replanning are reconsidered.

The execution model used in this chapter is intentionally lightweight. Its purpose is not to model detailed wheel–terrain interaction, actuator dynamics, or low-level tracking control. Instead, it provides a minimal closed-loop advancement model that makes it possible to evaluate sensing, path invalidation, replanning, and recovery under partial observability.

4.1.6 Backtracking and Recovery

When a failure condition is detected, either due to local minima or path invalidity, the agent initiates a recovery procedure. This involves returning to a previously safe state from which a new trajectory can be planned.

This mechanism is related to the recovery logic discussed in Chapter 3, but it is used here as an agent-side recovery procedure around the planner rather than as a new modification to the planner itself. By reverting to a known collision-free configuration, the agent avoids repeatedly attempting to replan from the same problematic state under limited environmental knowledge.

Following backtracking, the planner is re-invoked using the updated map, allowing the agent to continue navigation toward the goal.

4.2 Experimental Setup

The sensing-based navigation system was evaluated in five procedurally generated lunar-like environment scenarios, referred to here as Scenarios 1–5. Each scenario represents a different obstacle configuration and level of navigational difficulty.

To enable a controlled comparison, all experiments were conducted using two planner variants: Tuned RAPF and I-RAPF. Tuned RAPF denotes the baseline RAPF formulation using the optimized parameter set from Chapter 3. I-RAPF denotes the finalized improved planner from Chapter 3, using the same optimized parameter set together with the proposed algorithmic modifications. Both planners were evaluated under identical sensing conditions and across the same set of scenarios and random seeds.

For each scenario, 500 simulation episodes were executed using different random seeds, resulting in a total of 2500 runs per planner variant.

At the start of each run, the agent has no prior knowledge of the obstacle map. Obstacle information is acquired incrementally through local sensing during execution. The agent repeatedly senses, validates, replans, and advances until either the goal is reached or the run is terminated.

A run is considered *successful* when the agent reaches the goal region within the allowed simulation horizon and without collision. A run is considered *unsuccessful* when one of the following occurs:

1. the agent collides with an obstacle,
2. the maximum number of simulation steps is exceeded before the goal is reached,
3. the planner can no longer produce a viable continuation toward the goal.

The aim of this benchmark is not to evaluate low-level motion realism. Rather, it is to assess whether the RAPF-family planner can function reliably as the planning component of a sensing-driven navigation loop under partial observability, and to quantify the impact of the I-RAPF algorithmic improvements under identical experimental conditions.

4.3 Evaluation Metrics

For each simulation run, a set of metrics is recorded to characterize the performance of the navigation system under sensing-based conditions. These metrics are selected to capture reliability, planning efficiency, and trajectory quality. Table 4.1 summarizes the metric units.

Table 4.1: Evaluation metrics used for the sensing-based navigation experiments.

Metric	**Unit**	**Interpretation**
Success rate	%	Fraction of runs that reach the goal
Path length	m	Executed trajectory length
Planning effort	iterations	Cumulative planning iterations during a run
Replanning frequency	plan calls	Number of times the planner is invoked
Virtual obstacle usage	count	Number of unique virtual obstacles generated

Success rate. The primary metric is the fraction of runs in which the agent successfully reaches the goal. It is reported as a percentage (%). This metric reflects the robustness of the planner under partial observability and indicates its ability to handle local minima and dynamically discovered obstacles.

Path length. The executed path length provides a measure of trajectory efficiency and is reported in metres (m). It is computed as the cumulative Euclidean distance travelled by the rover during execution. Longer paths may indicate detours caused by replanning or inefficient navigation behaviour, while shorter paths suggest more direct and stable motion toward the goal.

Planning effort. The total planning effort is defined as the cumulative number of planning iterations performed during a run and is reported in planner iterations. This metric is dimensionless but represents computational work, as each iteration corresponds to one candidate expansion or update step inside the planner. It captures the computational cost of the navigation process and reflects how efficiently the planner converges to feasible solutions.

Replanning frequency. The number of plan calls corresponds to the number of replanning events triggered during execution and is reported as a count of planner invocations. Under partial observability, this metric provides insight into how often newly perceived obstacles invalidate the current trajectory and require replanning.

Artificial obstacle usage. The number of unique artificial obstacles encountered during a run is reported as a count. It is recorded as an indicator of how often the planner identifies and resolves local minima. Unlike cumulative counters, this metric reflects distinct problematic regions rather than repeated counting across replanning cycles.

Together, these metrics provide a comprehensive evaluation of the sensing-based navigation system, capturing not only whether the goal is reached, but also how efficiently and reliably the planner operates in dynamically revealed environments.

4.4 Illustrative Navigation Examples

Before presenting aggregate benchmark results, it is useful to visualize representative runs. Figure 4.2 shows a successful navigation run. The agent starts without prior knowledge of the environment and incrementally discovers obstacles through local sensing, which are visualized as red obstacles in the figure. As new obstacles are detected, the current trajectory is continuously validated. When a newly perceived obstacle invalidates the remaining path, replanning is triggered, resulting in local deviations from the original trajectory.

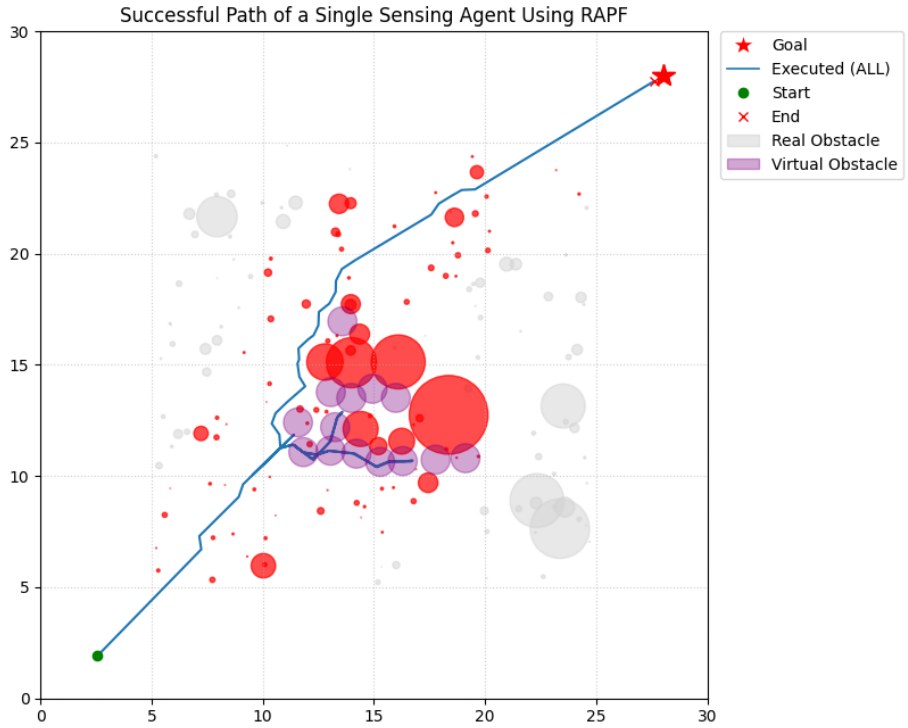

Figure 4.2: Representative successful navigation run in the sensing-based setting.

Despite these interruptions, the agent maintains consistent progress toward the goal. The resulting trajectory reflects a sequence of short planning horizons stitched together through repeated sensing and replanning. This behavior highlights the robustness of the navigation loop, demonstrating how feasible paths can be constructed online without global map knowledge.

4.5 Benchmark Results

This section presents the benchmark results for the five scenario classes, comparing Tuned RAPF with I-RAPF under identical sensing-based conditions. The results are reported in terms of success rate, planning efficiency, trajectory characteristics, failure modes, and artificial-obstacle usage. The interpretation of these results is discussed separately in Section 4.6.

4.5.1 Success Rate and Reliability

Table 4.2 presents the success rate of both planners across all scenarios.

I-RAPF achieved a higher success rate than Tuned RAPF in every scenario. Across all scenarios, I-RAPF reached an overall success rate of 99.84%, compared with 97.96% for Tuned RAPF. I-RAPF reached 100% success in Scenarios 1 and 3 and remained above 99% in the remaining scenarios. Tuned RAPF also performed well, but showed a larger number of unsuccessful runs across the full scenario set.

Expressed as failure probability, the difference is larger than the raw success rates suggest. Tuned RAPF failed in approximately 2.04% of runs, whereas I-RAPF failed in approximately 0.16% of runs. The number of unsuccessful episodes is therefore substantially reduced by the I-RAPF modifications.

Table 4.2: Success rate comparison per scenario, evaluated over 500 episodes per scenario.

Scenario	Tuned RAPF	I-RAPF
1	0.992	1.000
2	0.988	0.998
3	0.974	1.000
4	0.976	0.996
5	0.968	0.998

4.5.2 Planning Efficiency

Table 4.3 compares the planning efficiency of Tuned RAPF and I-RAPF for each sensing scenario. The table reports the mean value and one-standard-deviation spread over 500 episodes per scenario. The number of plan calls indicates how often the planner was invoked during an episode, while the planning effort represents the cumulative number of internal planning iterations required by those calls. Both quantities are dimensionless counts and are therefore reported with unit [-].

Table 4.3: Planning efficiency comparison per scenario. Values are reported as mean ± one standard deviation over 500 episodes per scenario.

Scenario	Plan Calls [-]		Planning Effort [-]	
	Tuned RAPF	I-RAPF	Tuned RAPF	I-RAPF
1	14.82 ± 8.74	9.34 ± 4.00	510.31 ± 236.30	358.81 ± 120.92
2	17.23 ± 10.01	10.95 ± 5.20	598.32 ± 284.70	405.44 ± 134.84
3	19.19 ± 11.64	11.49 ± 4.90	667.01 ± 345.19	429.78 ± 136.69
4	21.87 ± 11.88	12.64 ± 5.64	758.95 ± 360.81	469.63 ± 152.19
5	26.04 ± 14.81	14.31 ± 6.13	875.06 ± 454.69	508.86 ± 158.50

I-RAPF required fewer planner invocations in every scenario. Averaged over the scenario set, the number of plan calls decreased by approximately 40%. The cumulative planning effort was also lower for I-RAPF in every scenario, with an average reduction of more than 35%.

Both metrics increase with scenario complexity for both planner variants. This is expected because denser obstacle configurations create more path invalidations and therefore require more frequent replanning. However, I-RAPF maintains both a lower mean and a lower standard deviation across the full scenario set. This indicates that the improvement is not limited to average performance, but also reduces run-to-run variability in the sensing-based navigation loop.

The corresponding boxplots are provided in Appendix A.1. These plots show that the same trend is also visible in the median and interquartile range of the distributions.

4.5.3 Trajectory Characteristics

Table 4.4 presents the executed path length for both planner variants. Values are reported in metres as mean ± one standard deviation over 500 episodes per scenario. The path length is measured over the executed trajectory, rather than only over the planned waypoint sequence, and therefore reflects the combined effect of planning, replanning, and local navigation recovery.

Table 4.4: Executed path length comparison per scenario. Values are reported as mean ± one standard deviation over 500 episodes per scenario.

Scenario	**Tuned RAPF [m]**	**I-RAPF [m]**
1	44.53 ± 11.25	41.16 ± 5.93
2	44.85 ± 11.79	42.23 ± 8.19
3	45.36 ± 12.44	42.32 ± 7.84
4	46.60 ± 13.25	42.89 ± 8.47
5	49.01 ± 15.24	45.08 ± 10.25

I-RAPF produces shorter executed paths in every scenario. The difference is smaller than for the planning-efficiency metrics, but the trend is consistent across the scenario set. This suggests that the improved planner does not reduce planning effort by simply accepting longer or less direct trajectories. Instead, the lower planning demand is accompanied by a modest reduction in executed path length.

The standard deviation remains relatively large, especially for Tuned RAPF. This is caused by run-to-run variation in the amount of replanning and recovery required after new obstacles are detected. The appendix boxplots show that the path-length distributions contain long upper tails, which explains why the standard deviations are larger than the central box regions alone might suggest.

4.5.4 Failure Modes

Although both planner variants achieved high overall success rates, a small number of failures were observed. All recorded failures were timeout failures, despite also monitoring for collisions and backtrack-limit violations.

Timeout failures occur when the navigation loop repeatedly replans without making sufficient progress toward the goal within the allowed simulation horizon. In the observed timeout cases, the planner did not fail because of a direct obstacle collision. Instead, the agent became trapped in a cycle of replanning and local-minimum recovery.

A representative timeout case is shown in Figure 4.3. In this example, the agent explores multiple alternative routes, but each unsuccessful attempt inserts additional artificial obstacles. The resulting accumulation progressively restricts the available search space. In this specific run, 96 artificial obstacles were created before the run terminated.

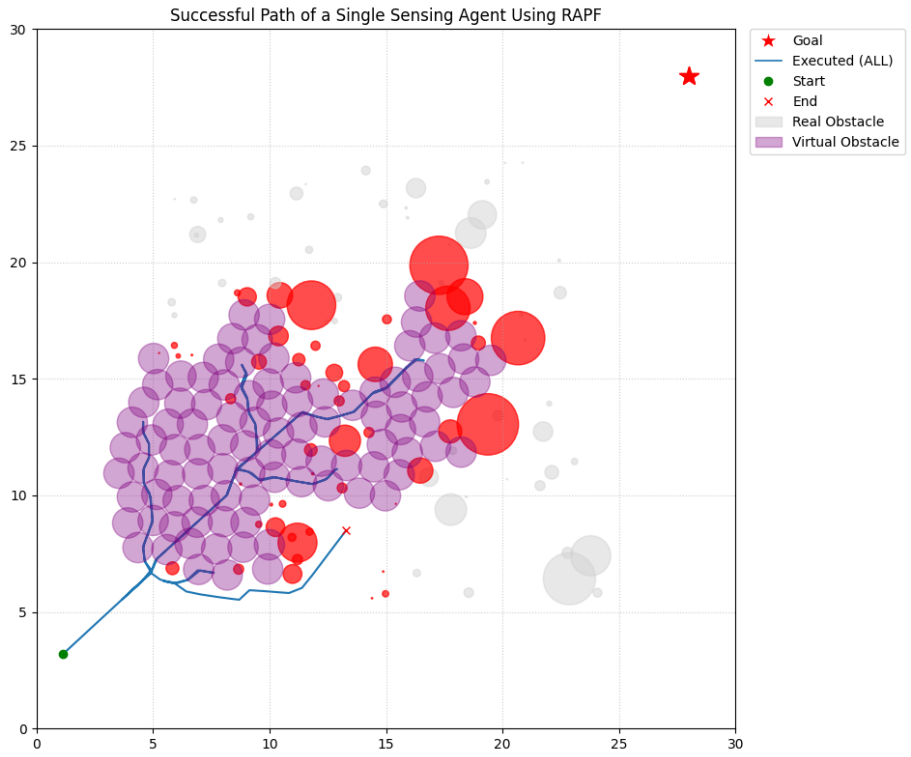

Figure 4.3: Timeout failure caused by accumulation of artificial obstacles, leading to a self-induced local minimum that prevents further progress using I-RAPF.

To further inspect this behaviour, a small exploratory experiment was conducted in which artificial obstacles were periodically cleared every N_{clear} replanning iterations. In a representative timeout scenario, this modification enabled the agent to escape the previously blocked region and successfully reach the goal. A qualitative illustration of this behaviour is provided in Appendix A.2.

4.5.5 Artificial Obstacle Usage

To examine the local-minimum recovery behaviour more closely, the number of unique artificial obstacles generated during each episode was recorded. Table 4.5 reports the average number of unique artificial obstacles per scenario.

Table 4.5: Average number of unique artificial obstacles per scenario.

Scenario	Tuned RAPF	I-RAPF
1	2.46	2.49
2	2.56	3.13
3	2.87	3.18
4	3.33	3.71
5	4.34	4.88

The mean number of unique artificial obstacles remains relatively low across all scenarios. The average count increases with scenario difficulty, which is consistent with denser environments producing more local-minimum situations and more replanning events.

However, the distribution of artificial-obstacle usage is not fully described by the mean. Figure 4.4 shows the distribution of artificial-obstacle counts for I-RAPF. Most runs use only a small number of artificial obstacles, while a small number of episodes produce much larger counts.

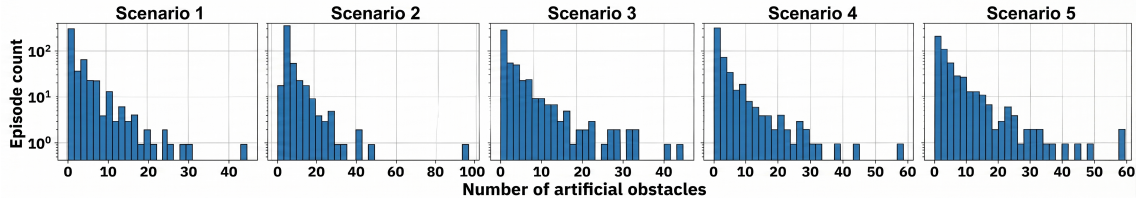

Figure 4.4: Histograms of the number of artificial obstacles per episode across scenarios using I-RAPF.

These high-count outliers are not exclusively associated with timeout failures, but they reflect the same stacking behaviour observed in the failure example. Repeated replanning can lead to continued insertion of artificial obstacles, which increases the difficulty of finding a feasible continuation in already constrained regions.

4.6 Discussion

The sensing-based benchmark shows that the improvements introduced in I-RAPF remain useful when the planner is embedded in a closed-loop navigation process. This is an important step beyond the full-map development setting of Chapter 3. In the present chapter, the rover does not start with complete knowledge of the environment. Instead, the active obstacle set is built incrementally during execution, and the current path may become invalid when newly sensed obstacles are added. The planner must therefore operate as part of a repeated loop of sensing, path validation, replanning, and execution.

The comparison between Tuned RAPF and I-RAPF indicates that the benefits of the algorithmic improvements are not limited to offline path generation. I-RAPF achieved a higher success rate, required fewer plan calls, reduced total planning effort, and produced shorter and less variable executed trajectories. These improvements are consistent with the design intent of the method. Midpoint collision checking improves the physical validity of planned segments, while partial-trajectory repair reduces unnecessary recomputation when local-minimum recovery is required. Under partial observability, this is especially valuable because replanning can be triggered repeatedly as new obstacles are discovered.

The planning-effort results are particularly important for the later swarm extension. In a single-agent setting, repeated replanning mainly affects the runtime of one rover. In a swarm setting, repeated replanning can occur across multiple agents and can interact with communication and coordination behaviour. Reducing unnecessary replanning in the single-agent planner therefore provides a stronger basis for the multi-agent architecture developed in the next chapters.

The results also show that the sensing-based setting introduces failure modes that are not visible in the same form under full map knowledge. In particular, rare timeout failures were associated with the accumulation of artificial obstacles. Artificial obstacles are useful because they prevent the planner from repeatedly returning to previously detected local-minimum regions. However, when many such obstacles accumulate in a confined region, they can progressively restrict the available search space. In extreme cases, the recovery mechanism can therefore create a self-induced local minimum: the artificial obstacles

inserted to escape earlier traps become part of a new blocking configuration.

This behaviour highlights a trade-off in persistent local-minimum memory. If artificial obstacles are removed too aggressively, the planner may revisit the same trap repeatedly. If they are retained indefinitely, the planner may over-constrain itself after many failed replanning attempts. The exploratory reset experiment discussed in Appendix A.2 suggests that controlled pruning, decay, or reset of artificial obstacles may improve robustness in these rare cases. However, such a mechanism was not integrated into the main planner in this thesis and therefore remains a topic for future work.

Another important interpretation is that the improved single-agent performance does not by itself solve the swarm navigation problem. The sensing-based I-RAPF loop demonstrates that one rover can use incrementally acquired obstacle information to maintain reliable progress toward the goal. In a swarm, however, each robot also becomes a moving constraint for the others, and a route that is feasible for one rover may not provide sufficient clearance for the group. The results of this chapter therefore provide the necessary planning foundation for the swarm architecture, but they do not yet address inter-agent coordination, shared route management, or formation-level route feasibility.

Overall, this chapter confirms that I-RAPF is a suitable planning component for the remainder of the thesis. It improves reliability and efficiency under partial observability while preserving the lightweight reactive character of RAPF. At the same time, the remaining artificial-obstacle accumulation failures and the absence of multi-agent interactions motivate the next stage of the work: extending the sensing-based planning framework toward swarm navigation.

4.7 Chapter Summary

This chapter evaluated I-RAPF in a sensing-based navigation loop. Unlike the controlled full-map experiments of Chapter 3, the rover did not start with complete obstacle knowledge. Instead, obstacles were detected online, added to the active obstacle set, and used to validate or invalidate the current path during execution.

The benchmark compared I-RAPF against Tuned RAPF under identical sensing-based conditions across the full scenario set. I-RAPF achieved higher goal-reaching reliability, required fewer replanning events, reduced total planning effort, and produced shorter and more consistent executed paths. These results show that the planner improvements introduced in Chapter 3 remain beneficial when planning and execution are interleaved under partial observability.

The chapter also identified a remaining limitation of the artificial-obstacle recovery mechanism. Although artificial obstacles usually help the planner escape local minima, excessive accumulation can occasionally over-constrain the search space and lead to timeout failures. This motivates future work on artificial-obstacle pruning, decay, or reset strategies.

The next chapter extends the navigation problem from a single sensing rover to a swarm of sensing and communicating rovers. This introduces new requirements that are not present in the single-agent setting, including shared obstacle discovery, inter-agent safety, coherent collective progress, and route compatibility with the spatial footprint of the group.

Chapter 5

Swarm Navigation Architecture and Formation-Aware Design

The previous chapter evaluated Improved RAPF (I-RAPF) in a sensing-based single-rover navigation loop. In that setting, the rover incrementally detected obstacles, updated its active obstacle set, validated the currently followed path, and replanned when newly detected obstacles invalidated the route. This established I-RAPF as a reactive planning component for one rover navigating through a partially known cluttered environment.

Extending this setting to a swarm changes the problem qualitatively. The objective is no longer only to guide one rover safely toward the goal, but to guide multiple robots through the same partially known environment while avoiding obstacles, avoiding one another, and maintaining collective progress. Even if each robot can independently generate a valid path to the goal, the swarm can still fail because agents interfere with one another, choose incompatible routes, or attempt to pass through regions that are feasible for a single rover but unsuitable for the spatial footprint of the group.

This chapter therefore introduces the swarm navigation problem and develops the architectural reasoning that leads to formation-aware route planning. The chapter does not yet define the final algorithmic implementation of the proposed stack. Instead, it identifies the additional requirements introduced by swarm navigation, compares candidate coordination approaches, and motivates why route-level formation reasoning is needed in addition to local coordination.

The central design argument is that reliable single-agent navigation does not automatically extend to reliable swarm navigation by running the same planner on every rover independently. I-RAPF provides each agent with a strong mechanism for sensing-based route generation and repair, but independently planned routes can still lead to conflicting motion decisions, uneven progress, or passages that are feasible for one rover but unsuitable for the group. The swarm architecture must therefore add a route-level mechanism that organizes how the agents use the available free space collectively. In this thesis, this structure is provided by a coordinator that maintains a formation-aware shared path, while local sensing, local coordination, and primitive-based execution remain distributed across the agents.

The chapter is structured as follows. Section 5.1 explains the transition from single-agent I-RAPF to swarm navigation. Section 5.2 defines the swarm problem formulation, including sensing, communication, active obstacle sets, and collective goal-reaching. Section 5.3 identifies the architectural requirements introduced by the swarm setting. Section 5.4 compares leader–follower and flocking-based coordination. Section 5.5 motivates formation-aware route planning. Section 5.6 presents the resulting layered architecture,

and Section 5.7 summarizes the runtime loop. Section 5.8 defines the Self-Planned Swarm baseline used later in the benchmark. This baseline represents the direct alternative of running I-RAPF independently on all agents, allowing the later comparison to isolate the value of coordinator-guided shared-route management. Section 5.9 states the assumptions and scope, and Section 5.10 summarizes the chapter.

5.1 From Single-Agent I-RAPF to Swarm Navigation

The sensing-based I-RAPF loop provides the starting point for the swarm formulation. In the single-agent case, the navigation system repeatedly senses nearby obstacles, updates the active obstacle set, checks whether the current path remains valid, and executes the next motion step. This structure remains useful in the swarm case, but additional couplings appear.

First, obstacle information is no longer limited to what one rover observes. Each agent can contribute detections to the navigation process, allowing the swarm to build a richer representation of the environment than any individual agent would obtain alone. Second, each robot becomes a moving constraint for the others. The navigation problem therefore includes both static environmental obstacles and dynamic inter-agent spacing constraints. Third, the group introduces a spatial footprint. A route that is safe for a single rover may not provide enough clearance for several rovers moving through the same region.

These changes mean that swarm navigation cannot be treated as a direct repetition of the single-agent planner for every robot. Independent planning may allow each agent to make progress, but it does not guarantee coherent collective motion. Conversely, adding a local coordination rule on top of a single-agent path may improve group behavior, but it cannot fully compensate for a route that is geometrically unsuitable for the swarm. The swarm extension therefore requires both shared information handling and a reconsideration of what it means for a route to be feasible.

5.2 Swarm Problem Formulation

Consider a swarm of N mobile robots operating in a two-dimensional environment representing a cluttered lunar surface. The index set of all agents is denoted by

$$\mathcal{I} = \{1, 2, \ldots, N\}. \tag{5.1}$$

Each agent $i \in \mathcal{I}$ has a position vector

$$\mathbf{p}_{i,k} = \begin{bmatrix} x_{i,k} \\ y_{i,k} \end{bmatrix} \in \mathbb{R}^{2\times 1}, \tag{5.2}$$

at discrete navigation step $k \in \mathbb{N}_0$. The mission-level goal is represented by the target position $\mathbf{g} \in \mathbb{R}^{2\times 1}$, and the goal region is defined by the radius R_{GOAL}. Static environmental obstacles, such as rocks and crater rims, are represented by the set $\mathcal{O}_{\mathrm{STAT}}$.

As in the sensing-based single-agent setting, the complete static obstacle set is not assumed to be known by the agents at the start of the episode. Instead, each agent maintains its own active obstacle set, denoted by $\mathcal{O}_{\mathrm{ACT},i}$. This set contains the obstacles currently known to agent i, including obstacles detected by the agent itself and obstacles received from other agents through communication. The set is updated during navigation and should therefore be interpreted as the current active obstacle set of agent i, rather than as a separate stored set for every time step.

At a given navigation step, agent i detects obstacles within its local sensing range. The latest detections made by agent i are denoted by

$$\mathcal{D}_i = \{o \in \mathcal{O}_{\text{STAT}} \mid \|\mathbf{p}_{i,k} - \mathbf{p}_o\|_2 \leq R_{\text{SENSE}}\}, \tag{5.3}$$

where $\mathbf{p}_o \in \mathbb{R}^{2\times 1}$ denotes the position of obstacle o, and R_{SENSE} is the sensing radius. The set $\mathcal{D}_i$ represents the result of the most recent sensing action of agent i. It is not a persistent map and it is not assumed to be globally available.

When an agent detects a previously unknown obstacle, the detection is added to its own active obstacle set and communicated to nearby agents. The communication neighborhood of agent i is denoted by

$$\mathcal{N}_i = \{j \in \mathcal{I} \mid j \neq i,\ \|\mathbf{p}_{j,k} - \mathbf{p}_{i,k}\|_2 \leq R_{\text{COMM}}\}, \tag{5.4}$$

where R_{COMM} is the communication radius. Communication is therefore local rather than global: a detection made by one agent is directly available only to agents within its communication neighborhood.

The active obstacle set of agent i is updated from two sources: its own most recent detections and the detections received from neighboring agents. This update can be written as

$$\mathcal{O}_{\text{ACT},i} \leftarrow \mathcal{O}_{\text{ACT},i} \cup \mathcal{D}_i \cup \bigcup_{j\in\mathcal{N}_i} \mathcal{D}_j. \tag{5.5}$$

This formulation deliberately avoids introducing a separate globally shared obstacle set. Obstacle knowledge is stored locally by each agent. Shared knowledge emerges through communication: when agents remain within communication range and exchange detections, their active obstacle sets may become more similar, but each agent still acts using its own $\mathcal{O}_{\text{ACT},i}$.

The route followed by the swarm is represented as an ordered waypoint sequence

$$\mathcal{P} = \{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_L\}, \qquad \mathbf{w}_\ell \in \mathbb{R}^{2\times 1}. \tag{5.6}$$

Here, $\mathbf{w}_\ell$ denotes waypoint ℓ in the path. The route is not assumed to remain valid throughout execution. As new obstacles are detected and communicated, the active obstacle information available to the swarm changes, and the remaining path may need to be validated, repaired, or replaced.

The swarm objective is to guide all agents toward the goal region while maintaining safety with respect to both environmental obstacles and other agents. Full-swarm convergence at navigation step k is achieved when

$$\|\mathbf{p}_{i,k} - \mathbf{g}\|_2 \leq R_{\text{GOAL}}, \qquad \forall i \in \mathcal{I}. \tag{5.7}$$

Inter-agent safety is represented by the minimum safety distance R_{SAFE}, requiring

$$\|\mathbf{p}_{i,k} - \mathbf{p}_{j,k}\|_2 > R_{\text{SAFE}}, \qquad \forall i, j \in \mathcal{I},\ i \neq j. \tag{5.8}$$

Obstacle safety is evaluated with respect to the obstacle information available to the relevant agent or planning layer. Since the agents do not necessarily have identical active obstacle sets, this creates an additional challenge: agents may act on partially synchronized environment knowledge.

This formulation defines the swarm navigation problem as a collective active-sensing problem with local communication. The environment is revealed incrementally through the detection sets $\mathcal{D}_i$, obstacle knowledge is maintained individually in $\mathcal{O}_{\text{ACT},i}$, and detections propagate through the swarm only through communication. The navigation architecture must therefore support collective progress under partially synchronized obstacle knowledge, while preventing both obstacle collisions and inter-agent conflicts.

5.3 Architectural Requirements for Swarm Navigation

Extending I-RAPF from a single rover to a swarm introduces requirements that are not present in the single-agent case. The architecture must not only find collision-free motion toward the goal, but must also support shared obstacle discovery, inter-agent safety, and coherent collective progress through cluttered terrain.

The first requirement is distributed obstacle awareness. Since each agent only senses obstacles within its local sensing radius, the environment is revealed incrementally and unevenly across the swarm. The navigation system must therefore allow locally detected obstacles, represented by the detection sets $\mathcal{D}_i$, to influence the behavior of other agents through communication. As defined in Section 5.2, these detections are incorporated into the local active obstacle sets $\mathcal{O}_{\mathrm{ACT},i}$. The architecture therefore does not rely on a globally known obstacle map, but on obstacle information that is gradually exchanged during navigation.

The second requirement is route maintenance under partial observability. A route that is valid when it is first generated may become invalid after new obstacles are detected. The system must therefore be able to validate the currently active route $\mathcal{P}$, repair it when possible, and replace it when necessary. This requirement follows directly from the sensing-based I-RAPF formulation, but becomes more important in the swarm setting because obstacle detections can originate from multiple agents.

The third requirement is inter-agent safety. Each robot is not only a navigating agent, but also a moving constraint for the rest of the swarm. The architecture must therefore prevent the agents from colliding with one another while they move toward the goal. This requirement cannot be handled by static obstacle avoidance alone, because the relative positions of the agents change continuously during execution.

The fourth requirement is route compatibility with the spatial footprint of the swarm. A path generated for a single rover may pass through regions that are collision-free for one robot but too narrow for a group moving together. In such cases, local coordination is forced to solve a problem created at the planning level. The route should therefore be evaluated not only with respect to individual obstacle clearance, but also with respect to the space required for coordinated swarm motion.

These requirements imply that the swarm architecture must combine route-level reasoning with local coordination. Route-level reasoning is needed to maintain a feasible path through the partially known environment. Local coordination is needed to regulate how the agents move relative to one another while executing that path. The following section compares two candidate coordination approaches and identifies the design implication for the final architecture.

5.4 Coordination Design Options

Two common coordination approaches are relevant for the swarm navigation problem considered in this thesis: leader–follower coordination and flocking-based local coordination. Both methods can produce coherent multi-agent motion, but they provide different forms of structure and impose different architectural costs.

5.4.1 Leader–Follower Coordination

Leader–follower coordination assigns one or more agents the role of leader, while the remaining agents regulate their motion relative to the leader or to a prescribed formation

frame. This gives the controller an explicit representation of the desired group structure: follower targets can be defined as relative positions, distances, or bearings with respect to the leader. In this sense, the method provides direct control over the intended formation geometry rather than relying only on emergent local interactions.

This structure is useful when the desired formation is known and should be maintained closely. For example, if the leader follows the planned route, the followers can be commanded to maintain offsets around the leader. More flexible variants have also been proposed, including dynamic leader selection as in Adderson et al. [12], and formation adaptation for navigation through constrained regions as in Yang et al. [26].

However, the explicit structure also introduces additional system requirements. A leader–follower architecture must define how leaders are selected, how follower references are assigned, how formation changes are triggered, and how agents recover when they fall behind or deviate from their assigned position. In cluttered environments, these recovery and reconfiguration mechanisms become central to the navigation system rather than minor implementation details. The method therefore offers strong structural control, but at the cost of increased coordination complexity.

5.4.2 Flocking-Based Local Coordination

Flocking-based coordination provides a more decentralized alternative. Instead of assigning fixed roles or formation slots, agents compute their motion using local interaction terms such as cohesion, alignment, and separation. This makes flocking attractive for swarm execution because it avoids explicit leader assignment and scales naturally with the number of agents [14].

In the context of this thesis, flocking is useful because the agents must react continuously to nearby robots and newly detected obstacles. A local interaction law can maintain group coherence while still allowing the swarm to deform when the environment becomes constrained. Behavior-based swarm methods follow a similar principle by combining goal attraction, obstacle avoidance, and inter-agent regulation into local control laws [15].

The limitation is that flocking does not directly encode route-level feasibility. The group shape is produced indirectly through local interaction terms, which means that the swarm may remain locally coherent while still spreading outside the clearance assumed by the planned route. Conversely, the swarm may compress or distort in narrow regions without any guarantee that the resulting motion remains compatible with the intended traversal corridor. Flocking therefore provides a flexible execution mechanism, but not a complete route-planning solution.

5.4.3 Comparison and Design Implications

Table 5.1 summarizes the relevant differences between the two coordination approaches for the present navigation problem.

Table 5.1: Comparison of candidate coordination approaches for swarm navigation in cluttered environments.

Criterion	**Leader–Follower Coordination**	**Flocking-Based Coordination**
Group structure	Explicitly represented through leader-relative references or a prescribed formation frame.	Emerges from local interaction terms such as cohesion, alignment, and separation.
Role assignment	Requires leader selection and follower reference assignment.	Does not require fixed leader or follower roles during local execution.
Formation maintenance	Can maintain prescribed relative positions when tracking errors remain small.	Maintains coherence indirectly through pairwise or neighborhood-level interaction rules.
Response to constrained passages	Can support planned reconfiguration, but requires explicit switching or recovery logic.	Can deform naturally, but deformation is not guaranteed to match route-level clearance assumptions.
Route-level feasibility	Does not by itself ensure that a route planned for one robot is feasible for the whole swarm.	Does not by itself ensure that the group remains inside the corridor assumed by the route.
Main architectural cost	Role management, reconfiguration, and recovery behavior.	Weaker direct control over the global swarm footprint.

The comparison shows that the two approaches have complementary strengths. Leader–follower coordination provides a more explicit representation of the intended group structure, but introduces role-management and recovery complexity. Flocking-based coordination is simpler to distribute and more flexible during execution, but the group structure is only indirectly controlled through local interaction terms.

More importantly, neither approach resolves the route-level problem by itself. Both methods can regulate how agents move relative to one another, but they do not determine whether the selected route provides enough space for the group to traverse it. If the route is generated from a single-agent perspective, the execution layer may be asked to move the swarm through a passage that is geometrically unsuitable for collective motion. This motivates shifting part of the swarm constraint from the coordination layer into the planning layer.

5.5 Need for Formation-Aware Route Planning

The coordination comparison shows that local coordination is necessary but not sufficient for robust swarm navigation in cluttered environments. Leader–follower and flocking-based methods can influence how agents move together, but they do not change the clearance properties of the route itself. A route that is too narrow for the group remains problematic regardless of the local coordination method used to execute it.

This distinction is important for obstacle-dense environments. When a route is planned using only the footprint of a single rover, it may pass close to obstacles or through narrow gaps. Such a route can still be valid for one robot, but difficult for several robots to execute simultaneously. The local coordination layer must then compensate for insufficient route

clearance by compressing the swarm, delaying agents, forcing agents away from the path, or triggering recovery behavior. These are execution-level responses to a planning-level limitation.

The planning layer should therefore evaluate route feasibility with respect to an effective swarm footprint. In this thesis, this footprint is represented by the formation radius $r_{\text{form}} > 0$, expressed in metres. The radius does not impose a rigid formation throughout execution. Instead, it defines the clearance requirement used when selecting, validating, and repairing a route. A larger value of r_{form} favors routes with more free space for coordinated motion, while a smaller value allows traversal through tighter regions when necessary.

This design choice is consistent with formation-aware and corridor-based planning ideas in the literature. Liu et al. [19] show that formation constraints reduce the feasible motion space and require reasoning about the multi-robot system rather than only individual agents. Sharma et al. [18] similarly highlight that route selection can depend on balancing preferred group configurations against alternatives that require less clearance. Tube- and corridor-based approaches also separate global spatial reasoning from local execution by first defining a traversable region and then constraining the swarm to move within it [27].

The resulting requirement is that route planning and local coordination must be coupled but not collapsed into a single layer. The route planner should determine where the swarm can move by accounting for the spatial footprint of the group. The local coordination layer should then determine how the agents move through that route while maintaining inter-agent safety and collective progress. This separation leads to the final architecture used in this thesis: formation-aware route management combined with tube-constrained local coordination and motion primitive execution.

5.6 Resulting Layered Swarm Navigation Architecture

The requirements and comparison above lead to a layered swarm navigation architecture. The central design choice is to separate route-level feasibility from local execution. The route management layer determines where the swarm should move, while the local coordination and motion execution layers determine how each agent moves through the selected route.

The resulting architecture consists of four functional layers: formation-aware route management, local coordination, motion execution, and communication. Table 5.2 summarizes these layers in terms of their role and output. This overview should be interpreted as an architectural decomposition rather than as the final algorithm specification. The concrete algorithms used to implement the route-management, local-coordination, and motion-execution layers are introduced in Chapter 6.

Table 5.2: Functional layers of the resulting swarm navigation architecture.

Layer	**Role**	**Output**
Formation-aware route management	Maintains and repairs a shared route while accounting for the effective swarm footprint.	Active waypoint route $\mathcal{P}$ and route updates.
Local coordination	Guides agents along the active route while regulating inter-agent spacing and collective motion.	Desired velocity $\mathbf{v}_{i,k}^{\text{des}}$.
Motion execution	Converts desired motion into feasible robot actions under non-holonomic and local safety constraints.	Executed motion primitive $\mathbf{u}_{i,k}$.
Communication	Shares obstacle detections, route updates, and agent state information.	Updated obstacle knowledge and shared navigation information.

The formation-aware route management layer is assigned to a coordinator. The coordinator maintains the active route $\mathcal{P}$, validates it when new obstacle information becomes available, and repairs or replans the route when necessary. Unlike the single-agent formulation, route validity is evaluated with respect to the formation radius r_{form}, so that the selected route reflects the clearance required by the swarm rather than only the footprint of one rover.

The local coordination layer executes the route in a distributed manner. Each agent uses the active route, neighboring-agent information, and local obstacle knowledge to compute a desired velocity $\mathbf{v}_{i,k}^{\text{des}}$. In the final architecture, this layer is implemented as a tube-constrained flocking controller. The flocking terms support cohesion, alignment, and separation, while the tube constraint links local motion to the corridor assumed by the formation-aware planner.

The motion execution layer converts the desired velocity into a feasible robot action. This layer is necessary because the rovers are non-holonomic systems and cannot directly execute arbitrary planar velocity commands. Each agent therefore evaluates candidate motion primitives and selects an executable action $\mathbf{u}_{i,k}$ that approximates the desired velocity while satisfying local safety checks.

The communication layer connects the individual agents and the coordinator. Obstacle detections are shared within communication range and incorporated into the local active obstacle sets defined in Section 5.2. The coordinator uses received detections to validate the shared route, while agents use route updates and neighboring-agent information during local execution. This structure allows the swarm to use shared obstacle information without assuming that all agents have an identical global map at all times.

5.7 Runtime Navigation Loop

Figure 5.1 summarizes the runtime loop of the proposed architecture. The notation in the diagram follows the problem formulation in Section 5.2: $\mathcal{D}_i$ denotes the latest detections of agent i, $\mathcal{O}_{\text{ACT},i}$ denotes its active obstacle set, and $\mathcal{P}$ denotes the active waypoint route.

The loop separates responsibilities between all agents and the coordinator. All agents

perform sensing, exchange newly detected obstacles within communication range, update their local obstacle information, and execute local motion. The coordinator additionally validates the shared route against newly available obstacle information. If the route remains valid, no route-level update is required and the agents continue executing the current route. If the route is invalidated, the coordinator repairs or replans the path and broadcasts the updated route.

The upper part of the loop corresponds to information gathering and route maintenance. Each agent senses obstacles in its local environment and shares newly detected obstacles with nearby agents. These detections update the local active obstacle sets $\mathcal{O}_{\mathrm{ACT},i}$. The coordinator uses the received obstacle information to check whether the current path $\mathcal{P}$ remains valid. This keeps the route reactive to newly discovered obstacles while avoiding unnecessary replanning when the current route is still feasible.

The lower part of the loop corresponds to local execution. Each agent either follows the current route or reconnects to it after a route update. The local coordination layer computes a desired motion command based on route tracking, tube adherence, and inter-agent regulation. The motion execution layer then selects a feasible motion primitive that respects the rover's motion constraints and local safety checks.

The goal-reaching check is shown outside the agent and coordinator lanes because it is treated as episode-level evaluation logic. In the experiments, success is defined by all agents satisfying the swarm goal condition from Eq. (5.7). The agents are therefore not required to run an explicit consensus protocol to determine whether the entire swarm has reached the goal.

This loop realizes the separation introduced in the layered architecture. The coordinator maintains a shared formation-aware route, while each agent performs local sensing, local coordination, and primitive-based execution. As a result, newly discovered obstacles can trigger route-level repair, while short-horizon motion remains distributed and reactive.

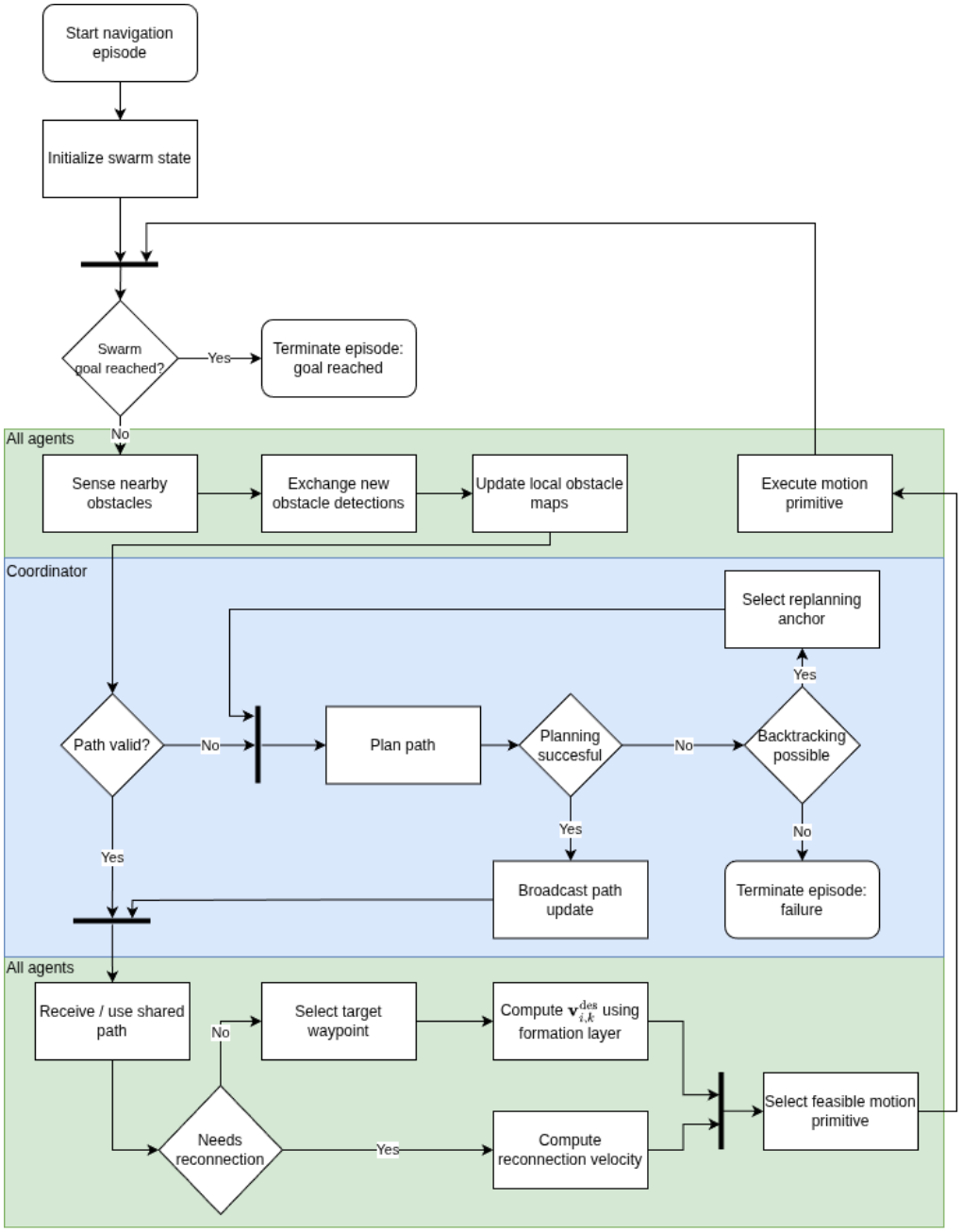

Figure 5.1: Runtime flow diagram of the coordinator–follower swarm navigation architecture.

5.8 Self-Planned Swarm Baseline

The proposed architecture is compared against a Self-Planned Swarm baseline, denoted SP-Swarm in the benchmark chapter. This baseline represents the most direct extension of the single-agent I-RAPF planner to a multi-robot setting: each rover uses the improved planner developed in Chapter 3 to plan and repair its own route toward the shared goal.

This comparison is important because I-RAPF performs strongly in the single-agent sensing-based setting. A natural alternative to the coordinator-guided architecture is therefore to give every agent the same improved planner and allow each rover to navigate in-

dependently through the environment. In this setup, the swarm behaviour emerges from multiple individually capable rovers operating at the same time, rather than from an explicitly maintained shared route.

In SP-Swarm, each agent maintains its own active obstacle set $O_{\mathrm{ACT},i}$, generates its own I-RAPF route, validates that route during execution, and replans when newly detected obstacles invalidate the current path. The agents pursue the same mission-level goal, but no common path P is maintained for the group. Route decisions are therefore made at the level of individual rovers rather than at the level of the swarm.

The baseline still uses the same sensing and execution assumptions as the proposed method wherever possible. Agents can detect obstacles locally, share newly detected obstacle information, and select feasible motion primitives using the same primitive execution layer. This ensures that the comparison does not simply favour the proposed method through different sensing or low-level actuation assumptions. The main difference is the presence or absence of coordinator-guided shared-route management.

SP-Swarm is therefore used to answer a specific question in the benchmark: if I-RAPF is already a reliable single-agent planner, is it sufficient to run it independently on all agents, or does the swarm require an additional route-level coordination structure? The benchmark in Chapter 7 evaluates this by comparing SP-Swarm against the proposed Coordinator-Guided Swarm stack, CG-Swarm. If independent I-RAPF execution were sufficient, SP-Swarm should achieve similar collective performance. If not, the performance gap indicates the value of planning and maintaining a shared formation-aware route for the swarm.

5.9 Assumptions and Scope

The swarm architecture is developed under the same discrete-time, two-dimensional navigation setting used throughout this thesis. The environment contains static obstacles such as rocks and crater-like terrain features. The mission-level goal $\mathbf{g}$ is known, and the episode succeeds when all agents reach the goal region.

Agents sense obstacles within a bounded radius R_{SENSE} and communicate within a bounded radius R_{COMM}. The architecture does not assume that all agents have identical obstacle knowledge at every navigation step. Instead, each agent maintains its own active obstacle set $\mathcal{O}_{\mathrm{ACT},i}$. These sets can become more similar through communication, but they are not replaced by a single globally shared obstacle set.

The coordinated architecture assumes that one agent or process acts as coordinator for route management. The coordinator maintains the shared path $\mathcal{P}$, validates it against newly available obstacle information, and broadcasts route updates after successful repair or replanning. Local execution remains distributed: each agent computes its own desired velocity and selects its own motion primitive.

The architecture focuses on navigation-level coordination rather than low-level hardware control. The motion primitive layer represents the interface between desired swarm motion and executable rover actions, but detailed actuator dynamics are outside the scope of this chapter. The objective is to evaluate whether formation-aware route management combined with local coordination improves swarm navigation in cluttered, partially known environments.

5.10 Chapter Summary

This chapter extended the navigation problem from a single sensing rover to a swarm of sensing and communicating rovers. The swarm setting introduces requirements that are not present in the single-agent case: distributed obstacle awareness, route maintenance under partial observability, inter-agent safety, coherent collective progress, and route compatibility with the spatial footprint of the group.

The chapter compared leader–follower coordination and flocking-based local coordination as candidate mechanisms for collective motion. Leader–follower coordination provides explicit group structure, but requires role management, reconfiguration, and recovery behaviour. Flocking-based coordination is easier to distribute and allows flexible local deformation, but does not directly guarantee that the swarm remains compatible with the clearance assumptions of the planned route.

This comparison motivated the use of formation-aware route planning. The final architecture separates route-level feasibility from local execution. A coordinator maintains and repairs a shared route using a formation-aware clearance requirement, while each agent performs local sensing, tube-constrained coordination, and motion primitive execution. A Self-Planned Swarm baseline was also defined to provide a direct comparison against the proposed coordinator-guided architecture. This baseline applies the improved single-agent planner independently on each rover, while keeping the sensing and primitive-execution assumptions as similar as possible. The later benchmark therefore evaluates whether reliable single-agent I-RAPF navigation is sufficient when applied to all agents, or whether swarm navigation benefits from an additional shared formation-aware route structure.

The next chapter builds on this architecture by defining the concrete algorithmic implementation of the proposed Coordinator-Guided Swarm stack. It describes how formation-radius reasoning is incorporated into I-RAPF, how the tube-flocking layer executes the resulting shared route, and how the motion primitive layer converts desired velocities into feasible rover actions.

Chapter 6

Layered Formation-Aware Swarm Navigation

The previous chapter established the architectural need for route-level swarm organization. A strong single-agent planner can provide each rover with a route toward the goal, but independent route generation does not guarantee coherent collective traversal. In cluttered environments, a route that is feasible for one rover may not provide enough clearance for the spatial footprint of the group, and local coordination may then be forced to compensate for a route-level limitation. Chapter 5 therefore motivated a coordinator-guided architecture in which the swarm follows a shared formation-aware route while local execution remains distributed.

This chapter defines the concrete algorithmic implementation of that architecture. The proposed Coordinator-Guided Swarm stack, denoted CG-Swarm, combines three coupled layers. First, a formation-aware I-RAPF planner generates a shared waypoint path and an associated formation-radius profile. Second, a tunnel-flocking controller converts this path and radius profile into desired velocities for the individual agents while maintaining path tracking, tube adherence, inter-agent spacing, and local obstacle avoidance. Third, a motion primitive layer converts these desired velocities into executable rover manoeuvres while enforcing local safety constraints.

The purpose of this chapter is to describe how the single-agent I-RAPF planner from Chapter 3 is extended into a swarm navigation method. The key extension is that route feasibility is no longer evaluated only for a single reference point. Instead, candidate route points are tested against an effective formation radius, allowing the planner to prefer wider routes where possible and compressed traversal where necessary. The resulting route representation consists of a path P and radius profile R_P, which together define both where the swarm should move and how much lateral space is available along the route.

The chapter also explains how this route-level representation is connected to executable agent motion. The tunnel-flocking layer treats the planned path as the centreline of a navigation corridor and uses the radius profile to regulate lateral deviation. The motion primitive layer then bridges the remaining gap between continuous desired velocities and the discrete, non-holonomic action constraints of the rover. The complete stack therefore links formation-aware route planning, local swarm coordination, and primitive-based execution.

The chapter is structured as follows. Section 6.1 gives an overview of the proposed CG-Swarm stack and its main algorithmic components. Section **??** presents the formation-aware I-RAPF planner. Section 6.3 describes the tunnel-flocking execution layer. Section 6.4 introduces the motion primitive layer. Section 6.5 describes the parameter tuning and implementation choices used for the final deployed stack. Section 6.6 discusses the de-

sign trade-offs of the layered method, and Section 6.7 summarizes the chapter and connects it to the benchmark evaluation.

6.1 Overview of the Proposed Method

The previous chapter defined the functional architecture required for swarm navigation: route management, local coordination, motion execution, and communication. This chapter defines the concrete algorithmic implementation of the route-management, coordination, and execution layers. The proposed method separates swarm navigation into planning, continuous coordination, and discrete execution. This separation is important because each layer addresses a different limitation. The planner determines whether the swarm can pass through the environment as a group. The flocking layer distributes the resulting route over the individual agents and maintains coherent group motion during execution. The primitive layer then ensures that the desired motion is compatible with the available robot actions and local collision constraints.

To make the later benchmark comparison reproducible, the formation-aware planner introduced in this chapter is denoted as *FA-I-RAPF*, short for *Formation-Aware Improved RAPF*. This name is used because the planner extends the improved RAPF formulation from Chapter 3 with formation-radius feasibility checks and radius-profile generation. The complete proposed closed-loop swarm navigation stack is denoted as *CG-Swarm*, short for *Coordinator-Guided Swarm*. In this stack, the coordinator maintains a shared formation-aware route for the group, while the tunnel-flocking controller and motion primitive layer execute that route at the agent level.

Table 6.1: Algorithmic components of the proposed CG-Swarm stack.

Layer name	**Component**	**Input**	**Output**
FA-I-RAPF	Formation-aware improved RAPF planner	Start position, goal position, active obstacle set $\mathcal{O}_{\mathrm{ACT},i_{\mathrm{coord}}}$, and candidate formation radii	Shared path $\mathcal{P}$ and radius profile $\mathcal{R}_{\mathcal{P}}$.
Tunnel-flocking controller	Continuous local coordination layer	Shared path $\mathcal{P}$, radius profile $\mathcal{R}_{\mathcal{P}}$, agent states, neighbouring agents, and local obstacles	Desired velocity $\mathbf{v}_{i,k}^{\mathrm{des}}$ for each agent i.
Motion primitive layer	Discrete execution layer	Desired velocity $\mathbf{v}_{i,k}^{\mathrm{des}}$, robot state, neighbouring agents, and local obstacles	Executable primitive $u_{i,k}$.

The complete CG-Swarm stack consists of the FA-I-RAPF planner, the tunnel-flocking controller, and the motion primitive layer under the coordinator–follower route-management structure introduced in Chapter 5. In the benchmark chapter, CG-Swarm refers to this

complete closed-loop stack, not to the formation-aware planner in isolation. This distinction is important because the observed swarm behaviour depends on the interaction between route planning, velocity-level coordination, and primitive-level execution.

The navigation stack is executed in a receding manner. The route is planned or repaired whenever the current shared path becomes invalid under the active obstacle set of the coordinating agent, $\mathcal{O}_{\mathrm{ACT},i_{\mathrm{coord}}}$, where $i_{\mathrm{coord}} \in \mathcal{I}$ denotes the route-managing agent. During execution, each agent evaluates its own state, nearby agents, and nearby obstacles to compute a desired velocity. This velocity is not applied directly. Instead, the motion primitive layer selects a feasible primitive from a finite primitive set, preventing the controller from requesting motions that cannot be realized by the platform.

Let $\mathcal{I} = \{1, \ldots, N\}$ denote the set of agent indices, where N is the swarm size. The position of agent $i \in \mathcal{I}$ at discrete navigation step k is denoted by $\mathbf{p}_{i,k} \in \mathbb{R}^{2\times 1}$. The formation-aware planner uses a route reference position $\mathbf{p}_k^{\mathrm{ref}} \in \mathbb{R}^{2\times 1}$, which is distinct from the physical agent positions $\mathbf{p}_{i,k}$. The shared route is denoted by $\mathcal{P} = \{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_M\}$, where each waypoint $\mathbf{w}_\ell \in \mathbb{R}^{2\times 1}$ describes a point on the planned route. The radius profile associated with this route is denoted by $\mathcal{R}_{\mathcal{P}} = \{r_{\mathrm{form},0}, r_{\mathrm{form},1}, \ldots, r_{\mathrm{form},M}\}$, where $r_{\mathrm{form},\ell} \in \mathbb{R}_{>0}$ defines the admissible formation radius near waypoint $\mathbf{w}_\ell$. The pair $(\mathcal{P}, \mathcal{R}_{\mathcal{P}})$ therefore describes both where the swarm should move and how much formation width is available along the path.

6.2 Formation-Aware I-RAPF Planner

6.2.1 Formation-Aware Extension of RAPF

The single-agent I-RAPF planner from Chapter 3 computes a collision-free path for one robot using the active obstacle set $\mathcal{O}_{\mathrm{ACT},i_{\mathrm{coord}}}$. For swarm navigation, this path is not sufficient. A route that is feasible for one robot may pass through a gap that cannot accommodate the formation. The formation-aware planner therefore extends the I-RAPF candidate evaluation by testing each candidate not only for the reference point of the path, but also for a formation footprint around that reference point.

The formation-aware extension follows the same general intuition as formation-layered planning approaches such as Sharma et al. [18], where route selection is influenced by whether the swarm can maintain a preferred formation or must temporarily switch to a narrower configuration. In contrast to graph-based multilayer formulations, the present method embeds this idea directly into the reactive RAPF candidate-selection process. Formation radii are not represented as separate graph layers; instead, each sampled RAPF candidate is tested against an ordered set of admissible radii and assigned the largest feasible radius.

The central idea is to associate each path segment with a formation radius. At each planning step, the planner considers candidate successor positions and evaluates whether the swarm can occupy a circular footprint of radius r_{form} around the candidate position without intersecting an obstacle. The candidate radius is selected from a finite ordered set

$$\mathcal{R}_{\mathrm{form}} = \{r_{\mathrm{form}}^{(1)}, r_{\mathrm{form}}^{(2)}, \ldots, r_{\mathrm{form}}^{(n_r)}\}, \qquad r_{\mathrm{form}}^{(1)} > r_{\mathrm{form}}^{(2)} > \cdots > r_{\mathrm{form}}^{(n_r)}, \tag{6.1}$$

where larger radii correspond to wider formation motion and smaller radii correspond to compressed motion through narrow passages. The planner attempts to preserve the largest feasible radius whenever possible, but can reduce the radius when the obstacle geometry requires a narrower formation.

In the implementation used for the benchmark experiments, this ordered set is chosen as

$$\mathcal{R}_{\text{form}} = \{1.0,\ 0.6,\ 0.3\}\ \text{m},$$

so that the planner first attempts to preserve a wide formation, then allows an intermediate formation, and finally permits a compact formation in narrow passages.

For a candidate position $\mathbf{c} \in \mathcal{C}_k$ and candidate formation radius $r_{\text{form}} \in \mathcal{R}_{\text{form}}$, feasibility is evaluated using a clearance condition with respect to the active obstacle set. For an obstacle $o \in \mathcal{O}_{\text{ACT},i_{\text{coord}}}$ with centre $\mathbf{p}_o \in \mathbb{R}^{2\times 1}$ and radius r_o, the candidate is formation-feasible if

$$\|\mathbf{c} - \mathbf{p}_o\|_2 > r_o + r_{\text{form}} + R_{SAFE}, \qquad \forall o \in \mathcal{O}_{\text{ACT},i_{\text{coord}}}, \tag{6.2}$$

where R_{SAFE} is the additional safety margin used by the planner. This condition turns obstacle clearance into a formation-level property. Instead of asking whether the reference point can pass the obstacle, the planner asks whether the formation footprint around the reference point can pass the obstacle. Equivalently, this condition can be interpreted as a surface-clearance test: the distance from the candidate point to the nearest obstacle boundary must exceed the tested formation radius plus the safety margin. In the implementation, an additional growth margin can be applied when switching back to a larger radius. This makes expansion more conservative than contraction and prevents the planner from immediately widening the formation after passing a locally narrow region.

6.2.2 Candidate Generation and Formation Bucketing

At each planning step k, I-RAPF generates a finite set of candidate successor positions around the current reference position $\mathbf{p}_k^{\text{ref}}$. This set is denoted by $\mathcal{C}_k \subset \mathbb{R}^{2\times 1}$. The formation-aware planner keeps this local candidate-generation structure, but augments each candidate with a feasible formation radius. The resulting set of formation-augmented candidate pairs is

$$\mathcal{B}_k = \{(\mathbf{c}, r_{\text{form}}^c) \mid \mathbf{c} \in \mathcal{C}_k\}, \tag{6.3}$$

where $\mathbf{c}$ is a candidate next reference position and r_{form}^c is the largest feasible formation radius assigned to that candidate.

The candidates are grouped into radius buckets. The highest bucket contains candidates that can support the largest formation radius. Lower buckets contain candidates that require a compressed formation. This creates a hierarchy in which spatially wider routes are preferred over narrow routes, provided that they still make progress toward the goal and satisfy the RAPF scoring criteria.

6.2.3 Hierarchical Reactive Selection

The planner selects the next path state using a hierarchical decision rule. Instead of scoring all candidates only by the RAPF potential value, the method first groups candidates by feasible formation radius and then applies the RAPF candidate score within each group. This preserves the reactive nature of RAPF while adding an explicit preference for routes that keep the formation wide.

Let $\mathcal{B}_k^{(q)} \subseteq \mathcal{B}_k$ denote the subset of augmented candidates feasible for radius $r_{\text{form}}^{(q)}$. The planner evaluates the buckets in descending radius order. The first non-empty bucket that contains candidates satisfying the progress, collision, and midpoint checks is selected.

Within that bucket, the candidate with the best RAPF score is chosen. This rule can be written as

$$\left(\mathbf{p}_{k+1}^{\mathrm{ref}}, r_{\mathrm{form},k+1}\right) = \underset{(\mathbf{c},r_{\mathrm{form}}^{c})\in\mathcal{B}_k^{(q^*)}}{\arg\min} j_{\mathrm{rapf}}\left(\mathbf{c}\right), \tag{6.4}$$

where j_{rapf} is the local RAPF successor score and q^* is the first feasible bucket in the ordered radius hierarchy.

This selection rule introduces a deliberate design choice. The planner does not always select the shortest or most attractive local step if that step forces the swarm into a smaller formation unnecessarily. Instead, it prioritizes maintaining a wider formation where the environment allows it, while still allowing compression when required to pass cluttered regions.

To avoid rapid switching between formation radii, the implementation uses asymmetric resize hysteresis. Shrinking to a smaller radius is allowed immediately, because this may be required to pass a narrow region. Expanding back to a larger radius is treated more conservatively: after a recent contraction, a short cooldown must expire before upsizing is permitted. This prevents the planner from alternating between wide and narrow radii in cluttered regions where the available clearance changes locally from one candidate step to the next.

6.2.4 Formation-Aware Planning Algorithm

Algorithm 3 summarizes the formation-aware planning procedure. The algorithm should be interpreted as the route-level extension of the I-RAPF planner introduced in Chapter 3. The same local candidate-sampling principle is retained, but each candidate is additionally tested for the largest formation radius that can be supported at that location. The output is therefore not only a waypoint sequence $\mathcal{P}$, but also a radius profile $\mathcal{R}_{\mathcal{P}}$ that is later used by the tunnel-flocking layer.

The helper function sampleCandidates$(\cdot)$ generates the local successor positions around the current route reference position, as in the single-agent RAPF formulation. The function formationClear$(\cdot)$ applies the formation-level clearance condition from Equation 6.2. This means that a candidate is rejected if the circular formation footprint around the reference position would intersect an obstacle or violate the safety margin. The candidate set $\mathcal{B}_k$ therefore contains only candidates that are feasible for at least one formation radius.

Algorithm 3 Formation-aware RAPF route planning

Input: $\mathbf{p}_0^{\text{ref}} \in \mathbb{R}^{2\times 1}$, $\mathbf{g} \in \mathbb{R}^{2\times 1}$, $\mathcal{O}_{\text{ACT},i_{\text{coord}}}$, $\mathcal{R}_{\text{form}}$
Output: Planned path $\mathcal{P}$ and formation-radius profile $\mathcal{R}_{\mathcal{P}}$
$\mathcal{P} \leftarrow [\mathbf{p}_0^{\text{ref}}]$
$\mathcal{R}_{\mathcal{P}} \leftarrow [\,]$
$k \leftarrow 0$
while $\|\mathbf{p}_k^{\text{ref}} - \mathbf{g}\|_2 > R_{GOAL}$ **do**
    $\mathcal{C}_k \leftarrow \text{sampleCandidates}(\mathbf{p}_k^{\text{ref}})$
    $\mathcal{B}_k \leftarrow \emptyset$
    for all $\mathbf{c} \in \mathcal{C}_k$ **do**
        $r_{\text{form}}^* \leftarrow \emptyset$
        for all $r_{\text{form}} \in \mathcal{R}_{\text{form}}$ **do**
            if $\text{formationClear}(\mathbf{c}, r_{\text{form}}, \mathcal{O}_{\text{ACT},i_{\text{coord}}})$ **then**
                $r_{\text{form}}^* \leftarrow r_{\text{form}}$
                break ▷ largest feasible radius is selected first
            end if
        end for
        if $r_{\text{form}}^* \neq \emptyset$ **then**
            add $(\mathbf{c}, r_{\text{form}}^*)$ to $\mathcal{B}_k$
        end if
    end for
    evaluate $j_{\text{rapf}}(\mathbf{c}, r_{\text{form}})$ for all $(\mathbf{c}, r_{\text{form}}) \in \mathcal{B}_k$
    $(\mathbf{c}_k^*, r_{\text{form},k}^*) \leftarrow \arg\min_{(\mathbf{c},r_{\text{form}})\in\mathcal{B}_k} j_{\text{rapf}}(\mathbf{c}, r_{\text{form}})$
    if $\mathcal{B}_k = \emptyset$ **then**
        handle blocked planning step ▷ repair, restart, or failure handling
    else
        $\mathbf{p}_{k+1}^{\text{ref}} \leftarrow \mathbf{c}_k^*$
        append $\mathbf{p}_{k+1}^{\text{ref}}$ to $\mathcal{P}$
        append $r_{\text{form},k}^*$ to $\mathcal{R}_{\mathcal{P}}$
        $k \leftarrow k + 1$
    end if
end while
return $\mathcal{P}$, $\mathcal{R}_{\mathcal{P}}$

The algorithm starts by initializing the path with the initial reference position $\mathbf{p}_0^{\text{ref}}$. At each planning step, a set of candidate successor positions $\mathcal{C}_k$ is sampled around the current reference position. For each candidate, the planner iterates through the ordered formation-radius set $\mathcal{R}_{\text{form}}$, starting with the largest radius. Once a feasible radius is found, the search over radii stops for that candidate. This implements the preference for wide formation motion without requiring all smaller radii to be tested unnecessarily.

After all feasible candidate-radius pairs have been collected, the planner evaluates the RAPF successor score j_{rapf}. This score retains the same role as in the I-RAPF planner: it selects a locally favourable successor based on goal attraction, obstacle avoidance, progress, and the additional checks introduced in Chapter 3, such as midpoint collision checking. The difference is that the candidate has now already passed the formation-clearance test. As a result, the selected successor $\mathbf{c}_k^*$ is not merely collision-free for the route reference point, but feasible for the associated formation radius $r_{\text{form},k}^*$.

If no candidate can support any formation radius, the planner treats the step as locally blocked. In the implementation, this is handled consistently with the I-RAPF repair mech-

anism from chapter 3: an artificial obstacle is inserted near the blocked reference position, and the partial path is rewound to a previous waypoint outside the influence region of that artificial obstacle. Planning then resumes from this earlier point using the updated active obstacle set. This preserves the recovery behaviour of I-RAPF while applying the stricter formation-clearance conditions introduced in this chapter.

6.2.5 Path Regularization and Zigzag Filtering

The formation-aware RAPF planner generates a path by repeatedly selecting candidate points from a discrete sampling ring around the current waypoint. Each candidate lies at approximately the same sampling distance from the current position. This discrete candidate structure is useful because it allows efficient local exploration of the configuration space, but it can also introduce a characteristic artifact in the raw path: short alternating direction changes, or zigzag oscillations.

These oscillations occur when consecutive planning steps produce lateral displacement with limited net forward progress. For example, the planner may move from waypoint $\mathbf{w}_i$ to $\mathbf{w}_{i+1}$ with a strong lateral component, and then select a subsequent waypoint $\mathbf{w}_{i+2}$ that partly cancels this lateral displacement. Although each individual step can locally reduce the RAPF objective, the combined motion over the two steps produces only limited advancement along the intended route.

This behaviour is an expected consequence of discrete reactive planning and does not necessarily indicate a failure of the potential field formulation. From a geometric planning perspective, the raw path may still be collision-free and feasible for the selected formation radius. However, it is undesirable for the downstream execution layers. In the proposed architecture, the tunnel-flocking controller interprets the planned path as the centreline of a moving corridor. Rapid local changes in path direction can therefore cause the agents to repeatedly cross the centreline, increasing lateral motion and making formation maintenance less stable.

Figure 6.1 illustrates this artifact. The left side shows a path generated by the formation-aware planner without the zigzag filter. Several consecutive waypoints alternate laterally, producing a visibly oscillatory centreline even though the route remains feasible. The right side shows the corresponding result after applying the filter, where unnecessary intermediate waypoints have been removed and the path becomes smoother for the downstream tunnel-flocking controller.

To reduce this effect, a lightweight path-regularization step is applied after planning. The filter examines triples of consecutive waypoints $(\mathbf{w}_i, \mathbf{w}_{i+1}, \mathbf{w}_{i+2})$. If the two consecutive segments form a short oscillatory pattern with limited net displacement relative to their combined length, the intermediate waypoint $\mathbf{w}_{i+1}$ is considered for removal. The waypoint is removed only if the direct segment from $\mathbf{w}_i$ to $\mathbf{w}_{i+2}$ remains valid.

For the formation-aware planner, this validity check must preserve both obstacle clearance and formation-radius feasibility. A shortcut between two non-adjacent waypoints is accepted only when the direct segment remains collision-free for the relevant formation radius along that part of the path. This prevents the regularization step from creating a path that is valid for the reference point but invalid for the formation footprint.

Importantly, the regularization step does not change the global planning strategy. It only removes local oscillatory artifacts from the already computed path. The resulting path therefore remains a formation-aware RAPF path, but provides a smoother directional guide for the tunnel-flocking controller.

The resulting path $\mathcal{P}$ and radius profile $\mathcal{R}_{\mathcal{P}}$ are then passed to the execution layer. The path describes the desired route of the swarm reference motion, while the radius profile

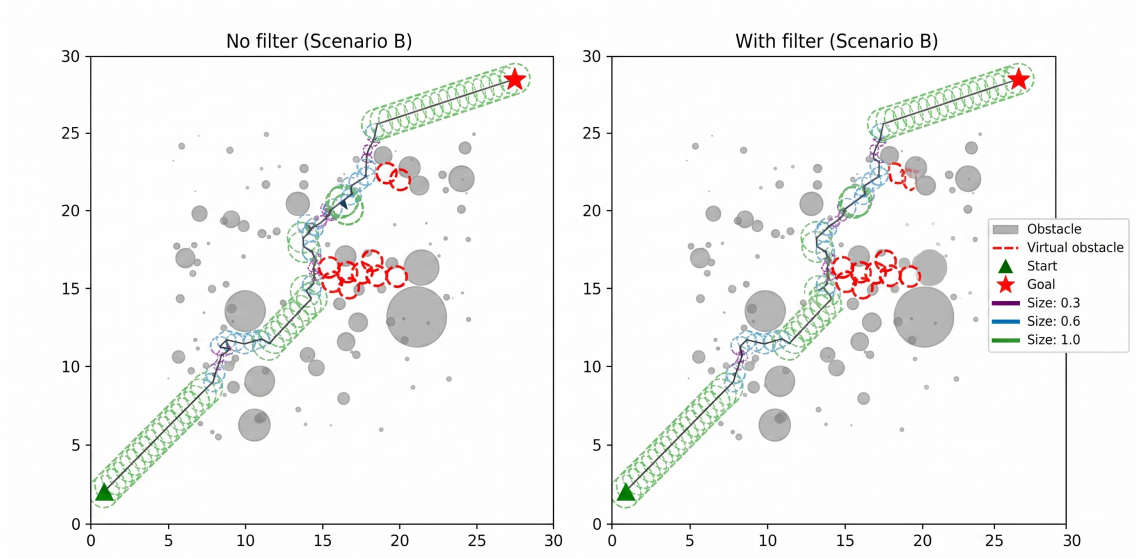

Figure 6.1: Example of zigzag artifacts in the raw formation-aware RAPF path and the filtered path used by the tunnel-flocking layer. The filtering step removes short alternating direction changes that would otherwise create unnecessary lateral motion in the swarm, while preserving obstacle clearance and formation-radius feasibility.

defines the allowed lateral spread along the route.

6.3 Tunnel-Flocking Execution Layer

6.3.1 Velocity Composition for Formation Path Tracking

During formation path tracking, each agent does not directly execute the geometric path produced by the formation-aware planner. Instead, the path is converted into a desired planar velocity, which is later passed to the motion primitive layer. This velocity construction follows the general flocking idea that group motion can be decomposed into local interaction terms and a task-directed navigation component [14]. It is also conceptually related to virtual-tube approaches, where a tube-like region is used to guide a swarm through cluttered environments while separating route-level spatial reasoning from local control [27]. In the present method, these ideas are used in a reactive form: the formation-aware RAPF planner provides a shared path and radius profile, and the tunnel-flocking layer converts them into a desired velocity composed of four terms: a path-following term, a neighbour-interaction term, a tube-correction term, and a soft obstacle-repulsion term. For agent i at discrete step k, the unclamped desired velocity is written as

$$\mathbf{v}_{i,k}^{\mathrm{raw}} = \mathbf{v}_{i,k}^{\mathrm{path}} + \mathbf{v}_{i,k}^{\mathrm{flock}} + \mathbf{v}_{i,k}^{\mathrm{tube}} + \mathbf{v}_{i,k}^{\mathrm{obs}}. \tag{6.5}$$

The term $\mathbf{v}_{i,k}^{\mathrm{path}}$ attracts the agent towards a look-ahead waypoint on the shared path. It therefore provides the primary forward motion along the planned route. The neighbour-interaction term $\mathbf{v}_{i,k}^{\mathrm{flock}}$ regulates the local behaviour between agents using alignment, cohesion, and separation. The tube-correction term $\mathbf{v}_{i,k}^{\mathrm{tube}}$ is only active when the agent moves outside the admissible tube around the current path segment. Finally, $\mathbf{v}_{i,k}^{\mathrm{obs}}$ gives a soft local repulsion from obstacles stored in the agent's local map. This term does not replace

the hard collision checks in the motion primitive layer, but biases the desired velocity away from nearby obstacles.

After these terms are summed, the resulting velocity is saturated by the maximum agent speed:

$$\mathbf{v}_{i,k}^{\mathrm{des}} = \begin{cases} \mathbf{v}_{i,k}^{\mathrm{raw}}, & \|\mathbf{v}_{i,k}^{\mathrm{raw}}\|_2 \le V_{\max}, \\ V_{\max} \dfrac{\mathbf{v}_{i,k}^{\mathrm{raw}}}{\|\mathbf{v}_{i,k}^{\mathrm{raw}}\|_2}, & \|\mathbf{v}_{i,k}^{\mathrm{raw}}\|_2 > V_{\max}. \end{cases} \tag{6.6}$$

This velocity is still not necessarily executed directly. In the experiments with motion primitives enabled, $\mathbf{v}_{i,k}^{\mathrm{des}}$ is used as the reference velocity for selecting a feasible primitive. The selected primitive must satisfy the local obstacle and neighbour safety checks and is therefore the final executed motion.

6.3.2 Per-Agent Path Tracking

Each agent tracks its own progress along the shared path rather than following a single global swarm center. When a new path is received, the agent initializes its progress by projecting its current position onto the path. During execution, this progress index is updated conservatively, after which a look-ahead point on the path is selected as the local reference.

This per-agent tracking is important because agents may temporarily lag behind one another due to local interactions, obstacle avoidance, or primitive-level constraints. By maintaining an individual path progress estimate, each agent can continue following the same shared route without requiring all agents to occupy the same path location at the same time.

The path-following component of the desired velocity is then constructed as a vector from the agent position toward this local look-ahead reference. Let $\mathbf{p}_{i,k}$ denote the position of agent i at time step k, and let $\mathbf{w}_{i,k}^{\mathrm{ref}}$ denote the selected look-ahead point on the shared path. The normalized path-tracking direction is

$$\mathbf{v}_{i,k}^{\mathrm{path}} = K_{\mathrm{path}} \frac{\mathbf{w}_{i,k}^{\mathrm{ref}} - \mathbf{p}_{i,k}}{\left\| \mathbf{w}_{i,k}^{\mathrm{ref}} - \mathbf{p}_{i,k} \right\|_2}. \tag{6.7}$$

This term gives all agents a common direction of travel while still allowing their individual positions to differ. It therefore acts as the connection between the global route and the local swarm motion.

6.3.3 Tube-Correction Velocity

The tube-correction term keeps agents inside the locally admissible formation tube. The word tube is used here in a practical execution-layer sense. Unlike optimal virtual-tube methods [27], which explicitly construct a tube containing optimal trajectories for the swarm, the tube in this thesis is derived directly from the formation-aware RAPF output. The path $\mathcal{P}$ acts as the centreline and the radius profile $\mathcal{R}_{\mathcal{P}}$ defines the locally admissible lateral deviation around that centreline. For the active path segment between waypoints $\mathbf{w}_m$ and $\mathbf{w}_{m+1}$, the closest point on the segment is computed by projecting the agent position onto the segment:

$$\tau_{i,k} = \mathrm{clip}\left(\frac{(\mathbf{p}_{i,k} - \mathbf{w}_m)^\top (\mathbf{w}_{m+1} - \mathbf{w}_m)}{\|\mathbf{w}_{m+1} - \mathbf{w}_m\|_2^2}, 0, 1 \right), \tag{6.8}$$

$$\mathbf{c}_{i,k} = \mathbf{w}_m + \tau_{i,k}(\mathbf{w}_{m+1} - \mathbf{w}_m). \tag{6.9}$$

The lateral offset from the path segment is

$$\mathbf{o}_{i,k} = \mathbf{p}_{i,k} - \mathbf{c}_{i,k}, \qquad d_{i,k}^{\text{tube}} = \|\mathbf{o}_{i,k}\|_2. \tag{6.10}$$

The local tube radius is taken as the smaller of the two formation radii associated with the segment endpoints:

$$r_m^{\text{tube}} = \min(r_m, r_{m+1}). \tag{6.11}$$

With deadband D_{tube}, the tube correction is only active when the agent lies outside $r_m^{\text{tube}} + D_{\text{tube}}$:

$$\mathbf{v}_{i,k}^{\text{tube}} = \begin{cases} \mathbf{0}, & d_{i,k}^{\text{tube}} \le r_m^{\text{tube}} + D_{\text{tube}}, \\ -k_{\text{tube}} \left(d_{i,k}^{\text{tube}} - r_m^{\text{tube}} - D_{\text{tube}} \right) \dfrac{\mathbf{o}_{i,k}}{d_{i,k}^{\text{tube}}}, & d_{i,k}^{\text{tube}} > r_m^{\text{tube}} + D_{\text{tube}}. \end{cases} \tag{6.12}$$

This correction is capped by a maximum tube-correction speed. In addition, the implementation disables the tube-correction term close to the final goal, so that agents are not forced to remain inside a narrow tube while settling around the terminal region.

6.3.4 Neighbour-Interaction Velocity

The neighbour-interaction term is the only part of the controller that is referred to here as the flocking velocity. For each agent, two local neighbour sets are constructed. The first set, $\mathcal{N}_i^{\text{int}}$, contains the nearest neighbours within an interaction radius and is capped by a maximum number of neighbours. This set is used for alignment and cohesion. The second set, $\mathcal{N}_i^{\text{sep}}$, contains neighbours within the separation radius and is used for collision-spacing behaviour. The two sets are separated because an agent may need to avoid close neighbours even when they are not part of the capped alignment and cohesion set.

This term follows the standard flocking interpretation of local alignment, cohesion, and separation, but it is used here only as a local execution mechanism inside the planned corridor. The route itself is still provided by the formation-aware planner, while the neighbour-interaction velocity regulates the relative motion of nearby agents during path tracking [**?**].

The alignment component steers the agent towards the mean velocity of the interaction neighbours:

$$\mathbf{v}_{i,k}^{\text{align}} = \frac{1}{|\mathcal{N}_i^{\text{int}}|} \sum_{j \in \mathcal{N}_i^{\text{int}}} \mathbf{v}_{j,k} - \mathbf{v}_{i,k}. \tag{6.13}$$

The cohesion component points from the agent towards the mean position of the same interaction neighbours:

$$\mathbf{v}_{i,k}^{\text{coh}} = \frac{1}{|\mathcal{N}_i^{\text{int}}|} \sum_{j \in \mathcal{N}_i^{\text{int}}} \mathbf{p}_{j,k} - \mathbf{p}_{i,k}. \tag{6.14}$$

The separation component is computed over the close-neighbour set. For each neighbour j, let

$$d_{ij,k} = \|\mathbf{p}_{j,k} - \mathbf{p}_{i,k}\|_2, \qquad \hat{\mathbf{e}}_{ij,k} = \frac{\mathbf{p}_{i,k} - \mathbf{p}_{j,k}}{d_{ij,k}}. \tag{6.15}$$

The contribution of neighbour j is then

$$\mathbf{s}_{ij,k} = \begin{cases} K_{\text{hard}} \dfrac{R_{\text{hard}} - d_{ij,k}}{d_{ij,k}} \hat{\mathbf{e}}_{ij,k}, & d_{ij,k} < R_{\text{hard}}, \\ \dfrac{r_{\text{sep}} - d_{ij,k}}{r_{\text{sep}}} \hat{\mathbf{e}}_{ij,k}, & R_{\text{hard}} \le d_{ij,k} < r_{\text{sep}}, \\ \mathbf{0}, & d_{ij,k} \ge r_{\text{sep}}. \end{cases} \tag{6.16}$$

The full separation velocity is

$$\mathbf{v}_{i,k}^{\text{sep}} = \sum_{j \in \mathcal{N}_i^{\text{sep}}} \mathbf{s}_{ij,k}. \tag{6.17}$$

The final neighbour-interaction velocity is then

$$\mathbf{v}_{i,k}^{\text{flock}} = k_{\text{align}} \mathbf{v}_{i,k}^{\text{align}} + k_{\text{coh}} \mathbf{v}_{i,k}^{\text{coh}} + k_{\text{sep}} \mathbf{v}_{i,k}^{\text{sep}}. \tag{6.18}$$

If no suitable neighbours are present, this term is zero and the agent follows the path using only the path, tube, and obstacle terms.

6.3.5 Local Obstacle Repulsion

Although the planner uses $\mathcal{O}_{\text{ACT},i_{\text{coord}}}$, local obstacle repulsion is still required during execution. This term compensates for tracking deviations, discretization effects, and newly detected obstacles that may not yet have triggered a full route repair.

Let $\mathcal{O}_{\text{LOC},i} \subseteq \mathcal{O}_{\text{ACT},i}$ denote the local obstacle set considered by agent i at the current navigation step. The obstacle repulsion term is defined as

$$\mathbf{v}_{i,k}^{\text{obs}} = K_{\text{OBS}} \sum_{o \in \mathcal{O}_{\text{LOC},i}} \psi_{\text{obs}} \left(d_{io,k}^{\text{obs}} \right) \frac{\mathbf{p}_{i,k} - \mathbf{p}_o}{\|\mathbf{p}_{i,k} - \mathbf{p}_o\|_2}, \tag{6.19}$$

where K_{OBS} is the obstacle-repulsion gain, $\mathbf{p}_o$ is the obstacle centre, and

$$d_{io,k}^{\text{obs}} = \|\mathbf{p}_{i,k} - \mathbf{p}_o\|_2 - r_o \tag{6.20}$$

is the clearance between agent i and obstacle o. The function $\psi_{\text{obs}}(\cdot)$ increases the repulsion as the clearance decreases.

6.3.6 Radius-Dependent Parameter Scheduling

The radius profile not only constrains the available corridor but also changes the preferred controller behaviour. In wide regions, the swarm can maintain a looser formation and rely more strongly on cohesion and alignment. In narrow regions, the controller must prioritize path tracking, tube adherence, and collision avoidance. For this reason, the deployed controller uses radius-dependent parameter scheduling.

Let $r_{i,k}^{\text{tube}}$ be the local radius obtained from $\mathcal{R}_{\mathcal{P}}$. The controller gains are selected as functions of this radius:

$$(K_{\text{PATH}}, K_{\text{TUBE}}, K_{\text{COH}}, K_{\text{SEP}}, K_{\text{ALI}}, K_{\text{OBS}}) = \gamma_{\text{sched}} \left(r_{i,k}^{\text{tube}} \right), \tag{6.21}$$

where $\gamma_{\text{sched}}(\cdot)$ denotes the gain-scheduling rule. This allows the same control structure to behave differently in wide and narrow route sections without changing the planner.

6.3.7 Tunnel-Flocking Velocity Procedure

Algorithm 4 summarizes the per-agent velocity computation used by the tunnel-flocking layer. Unlike the formation-aware planner, this algorithm is executed independently by each agent. Its role is to translate the shared route representation $(\mathcal{P}, \mathcal{R}_{\mathcal{P}})$ into a desired velocity $\mathbf{v}_{i,k}^{\text{des}}$ while also accounting for neighbouring agents and local obstacles.

The helper function selectPathReference$(\cdot)$ determines the current path reference for agent i, for example by selecting a nearby waypoint or a look-ahead point on $\mathcal{P}$. The function lookupTubeRadius$(\cdot)$ retrieves the local radius from $\mathcal{R}_{\mathcal{P}}$ corresponding to that part

of the path. The functions distanceToPath(·) and pathNormal(·) compute the geometric relation between the agent and the route centreline. Together, these quantities allow the controller to distinguish between motion along the path and motion away from the planned corridor.

Algorithm 4 Tunnel-flocking velocity computation

Input: agent state $\mathbf{x}_{i,k}$, path $\mathcal{P}$, radius profile $\mathcal{R}_{\mathcal{P}}$, neighbour set $\mathcal{N}_i$, local obstacle set $\mathcal{O}_{\mathrm{LOC},i}$
Output: Desired velocity $\mathbf{v}_{i,k}^{\mathrm{des}} \in \mathbb{R}^{2\times 1}$
$\mathbf{w}_{i,k}^{\mathrm{ref}} \leftarrow \mathrm{selectPathReference}(\mathbf{x}_{i,k}, \mathcal{P})$
$r_{i,k}^{\mathrm{tube}} \leftarrow \mathrm{lookupTubeRadius}(\mathbf{w}_{i,k}^{\mathrm{ref}}, \mathcal{R}_{\mathcal{P}})$
$d_{i,k}^{\mathrm{tube}} \leftarrow \mathrm{distanceToPath}(\mathbf{x}_{i,k}, \mathcal{P})$
$\mathbf{n}_{i,k} \leftarrow \mathrm{pathNormal}(\mathbf{x}_{i,k}, \mathcal{P})$
$\mathbf{v}_{i,k}^{\mathrm{path}} \leftarrow \mathrm{pathTrackingVelocity}(\mathbf{x}_{i,k}, \mathbf{w}_{i,k}^{\mathrm{ref}})$
$\mathbf{v}_{i,k}^{\mathrm{tube}} \leftarrow \mathrm{tubeCorrection}(d_{i,k}^{\mathrm{tube}}, r_{i,k}^{\mathrm{tube}}, \mathbf{n}_{i,k})$
$\mathbf{v}_{i,k}^{\mathrm{sep}} \leftarrow \mathbf{0}$
$\mathbf{v}_{i,k}^{\mathrm{coh}} \leftarrow \mathbf{0}$
$\mathbf{v}_{i,k}^{\mathrm{ali}} \leftarrow \mathbf{0}$
$\mathbf{v}_{i,k}^{\mathrm{obs}} \leftarrow \mathbf{0}$
for all $j \in \mathcal{N}_i$ **do**
    $\mathbf{v}_{i,k}^{\mathrm{sep}} \leftarrow \mathbf{v}_{i,k}^{\mathrm{sep}} + \mathrm{separationVelocity}(i, j)$
    $\mathbf{v}_{i,k}^{\mathrm{coh}} \leftarrow \mathbf{v}_{i,k}^{\mathrm{coh}} + \mathrm{cohesionVelocity}(i, j)$
    $\mathbf{v}_{i,k}^{\mathrm{ali}} \leftarrow \mathbf{v}_{i,k}^{\mathrm{ali}} + \mathrm{alignmentVelocity}(i, j)$
end for
for all $o \in \mathcal{O}_{\mathrm{LOC},i}$ **do**
    $\mathbf{v}_{i,k}^{\mathrm{obs}} \leftarrow \mathbf{v}_{i,k}^{\mathrm{obs}} + \mathrm{obstacleRepulsionVelocity}(i, o)$
end for
$\mathbf{v}_{i,k}^{\mathrm{flock}} \leftarrow \mathbf{v}_{i,k}^{\mathrm{sep}} + \mathbf{v}_{i,k}^{\mathrm{coh}} + \mathbf{v}_{i,k}^{\mathrm{ali}}$
$\mathbf{v}_{i,k}^{\mathrm{des}} \leftarrow \mathbf{v}_{i,k}^{\mathrm{path}} + \mathbf{v}_{i,k}^{\mathrm{tube}} + \mathbf{v}_{i,k}^{\mathrm{flock}} + \mathbf{v}_{i,k}^{\mathrm{obs}}$
$\mathbf{v}_{i,k}^{\mathrm{des}} \leftarrow \mathrm{limitVelocity}(\mathbf{v}_{i,k}^{\mathrm{des}}, V_{\mathrm{MAX}})$
return $\mathbf{v}_{i,k}^{\mathrm{des}}$

The first part of Algorithm 4 computes the path-related quantities required by the controller. The selected path reference $\mathbf{w}_{i,k}^{\mathrm{ref}}$ determines the direction in which the agent should progress. The local tube radius $r_{i,k}^{\mathrm{tube}}$ determines how much lateral deviation is allowed at that point along the route. The distance $d_{i,k}^{\mathrm{tube}}$ and normal vector $\mathbf{n}_{i,k}$ then define whether the agent should be corrected back toward the path centreline.

The second part computes the velocity contributions. The path-tracking term pulls the agent toward its path reference, while the tube-correction term prevents the agent from drifting outside the planned formation corridor. The separation, cohesion, and alignment terms are accumulated over the neighbour set $\mathcal{N}_i$. These terms are intentionally computed locally, because the flocking layer should not require centralized optimization over the full swarm state. Finally, local obstacle repulsion is accumulated over

6.3.8 Primitive Horizons and Effective Velocity

$\mathcal{O}_{\mathrm{LOC},i}$ to compensate for tracking errors, discretization effects, and newly sensed obstacles.

The resulting desired velocity is therefore a combination of route following, corridor enforcement, local swarm coordination, and reactive obstacle avoidance. The final call to limitVelocity(·) enforces the maximum speed constraint from Equation **??**. This is important because the velocity is later passed to the motion primitive layer, which can only approximate bounded local motion.

6.4 Motion Primitive Execution Layer

6.4.1 Purpose of the Motion Primitive Layer

The discrete action constraints of the Lunar Zebro rover were introduced in Section 2.3. There, the rover's executable low-level behaviour was represented by the elementary action set $\mathcal{A}$. In the swarm execution architecture, these elementary actions are not selected directly at every control step. Instead, they are grouped into a finite set of motion primitives, denoted by $\mathcal{U}$.

A motion primitive $u \in \mathcal{U}$ represents a short executable manoeuvre composed of one or more elementary actions from $\mathcal{A}$. The primitive set therefore forms an intermediate representation between the continuous desired velocity generated by the tunnel-flocking layer and the discrete non-holonomic action repertoire that the rover can execute.

The motion primitive layer acts as the interface between swarm-level velocity coordination and physically executable rover motion. Instead of applying the desired velocity directly, the controller selects the primitive whose predicted motion best approximates that desired velocity while satisfying local safety constraints.

This representation is useful for three reasons. First, it respects the rover's non-holonomic motion constraints without requiring the flocking layer to reason directly about low-level action sequences. Second, safety can be checked along the full rollout of a candidate manoeuvre rather than only at its endpoint. Third, the primitive set provides a practical way to approximate different effective speeds between agents, which is important for regulating spacing along the shared path.

When a new primitive is required at discrete time step k, agent i selects a primitive

$$u_{i,k} \in \mathcal{U}.$$

The selected primitive should approximate the desired velocity $\mathbf{v}_{i,k}^{\text{des}}$ while satisfying hard safety constraints with respect to obstacles and neighbouring agents.

6.4.2 Primitive Representation

A primitive $u \in \mathcal{U}$ is represented as a short sequence of elementary actions,

$$u = (a_0, a_1, \ldots, a_{m_u-1}), \qquad a_j \in \mathcal{A}, \qquad u \in \mathcal{U},$$

where m_u denotes the number of elementary actions contained in primitive u. Different primitives may therefore have different lengths, allowing $\mathcal{U}$ to contain both single-action primitives and short multi-action manoeuvres.

For agent i, applying primitive u at step k produces a predicted next pose

$$\hat{\mathbf{x}}_{i,k+1}(u) = f_{\text{prim}}\left(\mathbf{x}_{i,k}, u\right), \tag{6.22}$$

where $\mathbf{x}_{i,k}$ is the pose of agent i and $f_{\text{prim}}(\cdot)$ is the primitive transition model. The pose can include position and heading, for example

$$\mathbf{x}_{i,k} = \begin{bmatrix} x_{i,k} & y_{i,k} & \theta_{i,k} \end{bmatrix}^T. \tag{6.23}$$

The predicted displacement associated with primitive u is denoted by

$$\Delta\hat{\mathbf{p}}_{i,k}(u) = \hat{\mathbf{p}}_{i,k+1}(u) - \mathbf{p}_{i,k}. \tag{6.24}$$

6.4.3 Primitive Families

The primitive library used in this layer contains several families with different execution roles. These families are derived from the elementary action set $\mathcal{A}$, but they are not limited to single actions. In particular, the turning-related primitives are not pure in-place rotations. Instead, they are represented as short turn-and-forward sequences. In the implementation, such primitives may contain up to three left or right turning actions followed by forward motion.

This design choice is important for primitive selection. The scoring function compares candidate primitives using their effective velocity, heading alignment, and progress along the path. A primitive that only rotates in place would produce little or no translational displacement. It would therefore usually score poorly on the velocity-matching and path-progress terms, even when changing heading is the correct local behaviour. By including forward motion after the turning actions, the turning-related primitives remain comparable to forward and arc primitives: each normal candidate represents a short executable manoeuvre with both an orientation effect and a measurable displacement.

The forward primitive family contains primitives based mainly on the forward action. The arc family contains primitives based on arc_left and arc_right, allowing the rover to combine translation and heading change without explicitly stopping between actions. The recovery family contains primitives used only in more constrained local situations, such as backward motion or other fallback manoeuvres. These recovery primitives are penalized during selection to avoid unnecessary backward motion when a forward-safe alternative exists.

The primitive set can therefore be written abstractly as

$$\mathcal{U} = \mathcal{U}_{\text{align}} \cup \mathcal{U}_{\text{fwd}} \cup \mathcal{U}_{\text{arc}} \cup \mathcal{U}_{\text{rec}}, \tag{6.25}$$

where $\mathcal{U}_{\text{align}}$ contains turn-and-forward alignment primitives, $\mathcal{U}_{\text{fwd}}$ contains forward primitives, $\mathcal{U}_{\text{arc}}$ contains arc primitives, and $\mathcal{U}_{\text{rec}}$ contains recovery primitives.

6.4.4 Primitive Horizons and Effective Velocity

The flocking controller may require different agents to move with different effective velocities. For example, one agent may need to slow down to restore spacing, while another continues moving along the path. However, the primitive library consists of fixed geometric manoeuvres. The primitive layer therefore approximates velocity differences by evaluating primitives over their execution horizons.

Using the predicted displacement from Equation 6.24, the execution duration of primitive u is

$$T_u = n_u \Delta t, \tag{6.26}$$

where $n_u \in \mathbb{N}^+$ is the number of simulation steps covered by primitive u and Δt is the simulation time step. The effective velocity of primitive u is then approximated as

$$\mathbf{v}^{\text{eff}}_{i,k}(u) = \frac{\Delta\hat{\mathbf{p}}_{i,k}(u)}{T_u}. \tag{6.27}$$

This allows the selector to compare fixed motion primitives against the continuous desired velocity produced by the tunnel-flocking layer. A primitive with a similar geometric

displacement but a shorter execution duration corresponds to a higher effective velocity, while a longer execution duration corresponds to a lower effective velocity. In this way, agents can realize different effective speeds while still selecting from the same finite set of admissible primitive manoeuvres.

Some primitives may be committed for multiple low-level control steps once selected. During this interval, the agent follows the remaining rollout of the active primitive instead of selecting a new primitive at every simulation step. A new primitive is selected only when the active primitive has been completed or when the remaining rollout becomes unsafe due to newly sensed obstacles or nearby agents. This reduces rapid command switching and better reflects the finite duration of executable rover manoeuvres. It also requires conservative safety filtering, since each candidate primitive must be rejected if it violates obstacle or neighbour clearance anywhere along its predicted interval.

6.4.5 Collision Checking as a Hard Safety Filter

Primitive feasibility is evaluated before scoring. A primitive is rejected if its predicted motion intersects an obstacle or violates the minimum distance to a neighbouring agent. This ensures that safety is handled as a hard constraint rather than as a soft penalty in the objective.

For an obstacle $o \in \mathcal{O}_{\mathrm{LOC},i}$, a primitive is obstacle-feasible if the predicted swept motion maintains sufficient clearance:

$$\operatorname{dist}\left(\hat{\mathcal{S}}_{i,k}(u), o\right) > R_{\mathrm{ROBOT}} + R_{SAFE}, \qquad \forall o \in \mathcal{O}_{\mathrm{LOC},i}, \tag{6.28}$$

where $\hat{\mathcal{S}}_{i,k}(u)$ is the predicted swept set of agent i under primitive u, R_{ROBOT} is the robot radius, and R_{SAFE} is the safety margin.

For neighbouring agents, the predicted motion must also preserve inter-agent clearance:

$$\|\hat{\mathbf{p}}_{i,k+1}(u) - \mathbf{p}_{j,k}\|_2 > 2R_{\mathrm{ROBOT}} + R_{SAFE}, \qquad \forall j \in \mathcal{N}_i. \tag{6.29}$$

More conservative implementations may evaluate this condition over intermediate predicted samples along the primitive rather than only at the final predicted state.

The feasible primitive set is therefore

$$\mathcal{U}_{i,k}^{\mathrm{safe}} = \{u \in \mathcal{U} \mid u \text{ satisfies obstacle and neighbour feasibility}\}. \tag{6.30}$$

6.4.6 Primitive Selection

After filtering, each remaining primitive is scored according to how well it realizes the desired velocity and contributes to progress along the route. A generic primitive score is

$$j_{\mathrm{prim}}(u) = W_V \left\| \mathbf{v}_{i,k}^{\mathrm{eff}}(u) - \mathbf{v}_{i,k}^{\mathrm{des}} \right\|_2 + W_H e_\theta(u) - W_P \Delta s_i(u) + W_R \chi_{\mathrm{rec}}(u), \tag{6.31}$$

where W_V, W_H, W_P, and W_R are non-negative weighting constants. The first term penalizes the mismatch between the effective primitive velocity and the desired velocity from the tunnel-flocking layer. The term $e_\theta(u)$ penalizes heading misalignment, $\Delta s_i(u)$ rewards progress along the path direction, and $\chi_{\mathrm{rec}}(u)$ penalizes recovery primitives such as reverse motion.

For the standard forward, arc, and alignment primitives, each candidate includes a translational component. The velocity-matching and path-progress terms therefore compare manoeuvres with similar rollout semantics. This avoids the bias that would occur if pure in-place rotations were scored against translating primitives using the same

displacement-based objective. For this reason, the implemented primitive library represents alignment primitives as short sequences of at most three turning actions followed by forward motion, rather than as pure turning primitives.

The selected primitive is

$$u_{i,k}^{*} = \arg\min_{u \in \mathcal{U}_{i,k}^{\text{safe}}} j_{\text{prim}}(u). \tag{6.32}$$

If no safe primitive is available, the controller selects a conservative fallback action, such as waiting or turning away from the nearest constraint. This fallback prevents the agent from executing an unsafe motion when local space is temporarily blocked.

6.4.7 Motion Primitive Selection Procedure

In the implementation, primitive selection is only invoked when a new primitive is required. If an agent is already executing a committed primitive, the remaining rollout is first checked against the current local obstacle and neighbour information. If the active primitive remains safe, the agent continues executing it. If the primitive has been completed, or if the remaining rollout is no longer safe, a new primitive is selected using the procedure below. The algorithm therefore describes the selection subroutine rather than the complete low-level execution loop.

Algorithm 5 summarizes this primitive selection subroutine for a single agent. The algorithm has two stages. First, each candidate primitive is rolled out using the primitive transition model and filtered using hard safety constraints. Second, only the safe primitives are scored according to how well they approximate the desired velocity and contribute to progress along the path. This separation is important: collision avoidance is treated as a feasibility requirement, not as a soft term that can be traded against progress.

Algorithm 5 Motion primitive selection

Input: agent state $\mathbf{x}_{i,k}$, desired velocity $\mathbf{v}_{i,k}^{\text{des}}$, primitive set $\mathcal{U}$, neighbour set $\mathcal{N}_i$, local obstacle set $\mathcal{O}_{\text{LOC},i}$
Output: Selected primitive $u_{i,k} \in \mathcal{U}$
$\mathcal{U}_{i,k}^{\text{safe}} \leftarrow \emptyset$
for all $u \in \mathcal{U}$ **do**
    $\hat{\mathbf{x}}_{i,k+1}(u) \leftarrow f_{\text{prim}}(\mathbf{x}_{i,k}, u)$
    $\Delta\hat{\mathbf{p}}_{i,k}(u) \leftarrow \text{planarDisplacement}(\mathbf{x}_{i,k}, \hat{\mathbf{x}}_{i,k+1}(u))$
    $T_u \leftarrow \text{duration}(u)$
    $\mathbf{v}_{i,k}^{\text{eff}}(u) \leftarrow \Delta\hat{\mathbf{p}}_{i,k}(u)/T_u$
    $\hat{\mathcal{S}}_{i,k}(u) \leftarrow \text{sweptSet}(\mathbf{x}_{i,k}, \hat{\mathbf{x}}_{i,k+1}(u))$
    if $\text{collisionFree}(\hat{\mathcal{S}}_{i,k}(u), \mathcal{O}_{\text{LOC},i}, \mathcal{N}_i)$ **then**
        add u to $\mathcal{U}_{i,k}^{\text{safe}}$
    end if
end for
if $\mathcal{U}_{i,k}^{\text{safe}} = \emptyset$ **then**
    $u_{i,k} \leftarrow \text{waitPrimitive}()$ ▷ no feasible motion primitive
else
    evaluate $j_{\text{prim}}(u)$ for all $u \in \mathcal{U}_{i,k}^{\text{safe}}$
    $u_{i,k} \leftarrow \arg\min_{u \in \mathcal{U}_{i,k}^{\text{safe}}} j_{\text{prim}}(u)$
end if
return $u_{i,k}$

The algorithm begins by constructing the safe primitive set $\mathcal{U}_{i,k}^{\text{safe}}$. Each primitive $u \in \mathcal{U}$ is forward-simulated using $f_{\text{prim}}(\mathbf{x}_{i,k}, u)$, after which its swept set $\hat{\mathcal{S}}_{i,k}(u)$ is checked against the local obstacles and neighbouring agents. Unsafe primitives are discarded before scoring. This ensures that a primitive with good velocity tracking or path progress cannot be selected if it violates local clearance constraints.

If at least one safe primitive remains, the primitive score $j_{\text{prim}}(u)$ is evaluated over $\mathcal{U}_{i,k}^{\text{safe}}$. The selected primitive $u_{i,k}^{*}$ is the safe action that best approximates the desired velocity $\mathbf{v}_{i,k}^{\text{des}}$ while also encouraging progress along the route. If no safe primitive is available, a conservative fallback primitive is selected. This fallback behaviour prevents the agent from forcing motion in temporarily blocked local configurations and provides the final safety layer below the formation-aware planner and tunnel-flocking controller.

6.4.8 Relation to Swarm Coordination

The primitive layer is not only a low-level implementation detail. It affects swarm coordination because it determines which local motions are actually available to each agent. Without this layer, the flocking controller may request smooth velocity changes that are not executable by the rover. This mismatch can lead to delayed turns, overshoot, or local interference between agents. By selecting feasible short-horizon actions, the primitive layer reduces the gap between the velocity requested by the swarm controller and the motion executed by the platform.

The primitive layer also provides the final safety boundary in the layered architecture. The formation-aware planner checks route-level clearance, the tunnel-flocking layer coordinates local velocity commands inside the formation corridor, and the primitive layer checks the immediate executable motion before it is applied. This creates a layered safety structure: route-level feasibility, local velocity-level coordination, and action-level feasibility.

6.5 Parameter Tuning and Implementation Choices

The layered architecture contains several parameter groups, but not all of them were tuned independently. The formation-aware RAPF planner reuses the tuned I-RAPF parameters from Chapter 3, including the attractive and repulsive potential parameters, candidate-sampling settings, midpoint collision check, artificial-obstacle handling, and partial trajectory repair logic. This choice was made because the formation-aware extension does not introduce a new potential-field formulation. Instead, it changes the feasibility interpretation of each sampled candidate by testing whether a formation of radius r_{form} can safely occupy that candidate position.

The parameters that are specific to the formation-aware planner are therefore limited to the available formation radii, the ordering of those radii, and the resize logic used to prevent rapid switching between formation scales. The tunnel-flocking layer, on the other hand, introduces new execution-level parameters because it converts a planned path and radius profile into decentralized agent velocities. These flocking parameters were tuned separately, since they govern inter-agent spacing, tube tracking, along-path progress, and local collision avoidance during execution.

6.5.1 Frozen-Path Benchmark

Before evaluating the full swarm navigation stack, the velocity-level tunnel-flocking controller is tuned in a frozen-path benchmark. The purpose of this benchmark is to isolate

the execution behaviour of the controller from the additional effects of online sensing, route repair, and motion primitive selection. This is analogous to the planner tuning procedure described in Chapter 3, where candidate parameter sets are compared through repeated rollouts under controlled conditions. Here, however, the optimized component is not the path planner but the controller that converts a fixed route into swarm motion.

During this tuning setup, online sensing and route repair are disabled. The agents are preloaded with the obstacle map used to generate the frozen routes, so the local obstacle information remains fixed throughout the rollout. Each agent follows a precomputed private path, while pose reports are still exchanged to support local inter-agent interaction. The execution is performed in direct-velocity mode rather than through the motion primitive layer. The benchmark therefore isolates the velocity-level tunnel-flocking controller before the full sensing, replanning, coordination, and primitive-execution stack is evaluated.

6.5.2 Controller Optimization Objective

The controller parameters are optimized using Optuna. A candidate parameter vector $\boldsymbol{\theta}$ defines the gains and thresholds used by the velocity-level execution layer, including the path-following scale, separation gains, obstacle-repulsion parameters, obstacle speed cap, and goal stopping radius. For each trial, the candidate parameter vector is evaluated over a set of frozen-path rollout cases. The objective value assigned to the trial is the average score over these rollouts.

For one optimizer trial, the candidate controller parameter vector $\boldsymbol{\theta}$ is evaluated over N rollout cases. Each rollout i produces a scalar score s_i, and the trial objective is

$$j_{\text{trial}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} s_i. \tag{6.33}$$

The score for rollout i is defined as

$$\begin{aligned} s_i = {} & 1800\, N_{\text{obs},i} + 700\, N_{\text{agent},i} + 220\, (n_{\text{agents}} - n_{\text{reached},i}) \\ & + 25\, r_{\text{overspeed},i} + 5\, s_{\text{osc},i} + 0.12\, N_{\text{steps},i} - 140\, \mathbb{I}_{\text{success},i}. \end{aligned} \tag{6.34}$$

Here, $N_{\text{obs},i}$ is the accumulated number of obstacle collisions in rollout i, $N_{\text{agent},i}$ is the accumulated number of inter-agent collisions, n_{agents} is the number of agents in the rollout, and $n_{\text{reached},i}$ is the number of agents that reached the goal region. The term $r_{\text{overspeed},i}$ is the fraction of raw desired-velocity commands whose norm exceeded the maximum velocity before clamping, $s_{\text{osc},i}$ is the mean oscillation score computed from changes in executed velocity, and $N_{\text{steps},i}$ is the rollout duration in simulation steps. Finally, $\mathbb{I}_{\text{success},i}$ is an indicator that equals one when all agents reach the goal region and zero otherwise.

This objective should not be interpreted as a physical energy, distance, or time quantity. The terms combine quantities with different units and scales, but they are used here only as a dimensionless ranking criterion for comparing controller parameter sets under identical benchmark conditions. The weights therefore encode a priority ordering rather than a physical model. Obstacle collisions receive the largest penalty and dominate the score, because terrain contact represents a critical failure. Inter-agent collisions are also strongly penalized, since they indicate unsafe swarm interaction. Incomplete traversal is penalized through the number of agents that fail to reach the goal region. Overspeeding, oscillation, and rollout duration act as secondary criteria that distinguish between otherwise feasible or near-feasible executions. The negative success bonus further separates complete swarm traversal from partial completion.

Averaging the rollout scores in Equation 6.33 prevents the optimizer from selecting parameters that perform well on only a single route or scenario. Instead, the selected parameter vector must produce consistently safe and effective behaviour across the training cases.

6.5.3 Final Deployed Parameters

The final deployed parameters use the same controller structure across all scenarios, with radius-dependent scheduling where narrow passages require more structured path and tube tracking. The parameters should be interpreted as an optimization-derived baseline followed by a small implementation-level refinement.

During later validation, a bug was identified in the local obstacle-memory pipeline: already detected obstacles could be appended repeatedly to the local map. For the flocking controller, this artificially amplified the local obstacle repulsion term. After this issue was corrected, the optimized baseline remained broadly useful, but the smallest-radius regime required additional refinement to restore smooth flow through tight passages.

The wider radius classes therefore retain the optimized baseline more directly, while the smallest-radius parameter block places relatively more emphasis on structured forward progress and tube tracking. The final parameter values are reported together with the benchmark results in Chapter 7.

6.6 Discussion

The layered design addresses the main limitation identified in Chapter 5: local coordination alone can organize the relative motion of agents, but it cannot guarantee that the route itself is suitable for the swarm footprint. CG-Swarm addresses this limitation by moving part of the swarm constraint into the planning layer. The formation-aware planner encodes the available formation radius directly into the route, the tunnel-flocking layer uses this route as a corridor rather than as a single geometric line, and the primitive layer ensures that the desired local motion can be executed safely.

This separation also makes the method modular. The planner can be improved without changing the primitive set, the flocking controller can be tuned without redefining the global route representation, and the primitive layer can be adapted to different robot platforms. At the same time, the layers remain coupled through well-defined variables: the planner outputs $(\mathcal{P}, \mathcal{R}_{\mathcal{P}})$, the flocking layer outputs $\mathbf{v}_{i,k}^{\mathrm{des}}$, and the primitive layer outputs $u_{i,k}^{*}$.

The main trade-off is that each layer introduces its own approximation. The formation-aware planner represents the swarm footprint with a radius, the flocking layer represents group motion through local velocity terms, and the primitive layer approximates continuous desired motion through a finite action set. The benchmark chapter therefore evaluates the complete stack rather than only the planner, because the final behaviour depends on the interaction between all layers.

6.7 Chapter Summary

This chapter presented the concrete algorithmic implementation of the coordinator-guided swarm navigation architecture introduced in Chapter 5. The resulting CG-Swarm stack consists of a formation-aware I-RAPF planner, a tunnel-flocking execution layer, and a motion primitive execution layer.

The formation-aware planner extends I-RAPF by evaluating candidate route points with respect to an effective formation radius. It generates both a shared path P and a formation-radius profile R_P, allowing the route to represent not only where the swarm should move, but also how much lateral space is available along the path. The tunnel-flocking layer converts this route representation into desired velocities for each agent by combining path tracking, tube correction, neighbour interaction, and local obstacle repulsion. The motion primitive layer then converts these desired velocities into executable rover actions while applying hard safety filters for obstacles and neighbouring agents.

The chapter also described the tuning and implementation choices used for the final deployed stack. The formation-aware planner reuses the optimized I-RAPF parameters from Chapter 3, while the tunnel-flocking parameters were tuned in a frozen-path benchmark to isolate execution behaviour before evaluating the full sensing and replanning stack.

Together, these layers connect route-level formation feasibility to local executable motion. The next chapter evaluates the complete CG-Swarm stack in simulation and compares it against the Self-Planned Swarm baseline, SP-Swarm, to determine whether coordinator-guided shared-route management improves collective navigation beyond running I-RAPF independently on all agents.

Chapter 7

Swarm Benchmarking and Results

The previous chapters established the design rationale and implementation of the proposed swarm navigation system. Chapter 5 introduced the layered swarm navigation architecture and motivated the use of coordinator-based route management for cluttered multi-robot navigation. Chapter 6 then defined the concrete algorithmic stack used in this thesis: the formation-aware improved RAPF planner, denoted FA-I-RAPF, the tunnel-flocking controller, and the motion primitive layer.

This chapter evaluates *CG-Swarm* against the Self-Planned Swarm baseline, denoted *SP-Swarm*. SP-Swarm represents the direct alternative of running the improved single-agent I-RAPF planner independently on all agents while keeping the sensing and primitive-execution assumptions as similar as possible. The comparison therefore tests whether the single-agent planner developed earlier in the thesis is sufficient when applied to all rovers, or whether swarm navigation benefits from an additional shared formation-aware route structure.

The purpose of this evaluation is not only to determine whether the swarm reaches the goal, but also to examine how the two route-organization strategies affect collective convergence, planning effort, local traffic, inter-agent safety, communication behaviour, and motion efficiency. Since both stacks use the same underlying sensing assumptions and the same primitive execution layer, the benchmark focuses on the effect of coordinator-guided shared-route management relative to independently maintained individual routes.

The chapter is structured as follows. Section 7.1 describes the experimental setup, including the evaluated algorithmic stacks, scenario set, benchmark configuration, and illustrative successful benchmark snapshots. Section 7.2 defines the evaluation metrics used throughout the analysis. Section 7.3 presents the aggregate results across all scenarios, followed by a scenario-wise analysis in Section 7.4. Section 7.5 analyzes the observed failure categories. Section 7.6 interprets the main performance differences between the two evaluated stacks. Section 7.7 discusses communication and coordination behavior. Section 7.8 examines illustrative extreme cases that connect the quantitative benchmark results to observed rollout behavior. Section 7.9 presents preliminary scaling observations for larger swarm sizes. Finally, Section 7.10 summarizes the chapter.

7.1 Experimental Setup

7.1.1 Evaluated Algorithmic Stacks

Two closed-loop swarm navigation stacks were evaluated in the benchmark. Both stacks operate under the same sensing and communication assumptions: newly detected obstacles

are shared within communication range and incorporated into the agents' active obstacle knowledge. Both stacks also use the same motion primitive layer for executable robot actions. The stacks differ in how route planning and route organization are handled.

Table 7.1: Algorithmic stacks evaluated in the swarm benchmark.

Stack	**Algorithmic components**	**Route organization**
Coordinator-Guided Swarm (CG-Swarm)	FA-I-RAPF planner, tunnel-flocking controller, and motion primitive layer; see Algorithms 3, 4, and 5.	A coordinator maintains a shared formation-aware route $\mathcal{P}$ and radius profile $\mathcal{R}_{\mathcal{P}}$. Agents execute this shared reference through local coordination and primitive-based execution.
Self-Planned Swarm (SP-Swarm)	I-RAPF planner and motion primitive layer; see Chapter 3 and Algorithm 5.	Each agent maintains and repairs its own I-RAPF route. Agents share obstacle detections, but no common formation-aware route or radius profile is maintained.

The first evaluated stack, CG-Swarm, is the proposed method developed in Chapter 6. At the planning level, the FA-I-RAPF planner generates a shared path $\mathcal{P}$ together with a formation-radius profile $\mathcal{R}_{\mathcal{P}}$. At the coordination level, the tunnel-flocking controller converts this shared reference into a desired velocity $\mathbf{v}_{i,k}^{\text{des}}$ for each agent while regulating path progression, tube adherence, neighbor interactions, and obstacle repulsion. At the execution level, the motion primitive layer selects a feasible primitive $u_{i,k}$ from the available primitive set after applying hard safety checks.

The second evaluated stack, SP-Swarm, is the independent-planning baseline. This baseline follows the self-planning alternative discussed in Chapter 5. Agents exchange newly sensed obstacle information, but each agent maintains and repairs its own route using the single-agent I-RAPF planner from Chapter 3. The resulting desired local motion is then passed to the same motion primitive layer used by CG-Swarm.

The comparison therefore isolates the effect of route organization and coordinator-guided execution. CG-Swarm uses a shared formation-aware route with explicit coordinator-based route management, whereas SP-Swarm uses independently maintained routes for each agent. Since both stacks use shared obstacle information and the same primitive execution layer, performance differences are not attributed to different sensing assumptions or different low-level actuation constraints.

7.1.2 Benchmark Configuration

Each stack was evaluated across the five benchmark scenarios described in Section **??**, using multiple randomized seeds. Due to the substantially higher computational cost of swarm simulation compared to single-agent evaluation, the number of episodes per scenario was reduced relative to the single-agent benchmark. Nevertheless, the number of trials remained sufficient to support a structured comparison across algorithmic stacks and scenario classes.

Unless stated otherwise, the benchmark used the following default settings:

- 100 episodes per scenario per stack,
- 6 agents per rollout,
- a step horizon of 1800 simulation steps,
- a communication radius of 10.0 m,
- a sensing radius of 3.0 m,
- a goal radius of 4.0 m,
- and a robot radius of 0.20 m.

These settings reflect the benchmark configuration used in the evaluation script and provide a common basis for all reported comparisons.

7.1.3 Reproducibility and Logging

For each rollout, the benchmark recorded a structured set of metrics describing outcome, motion, planning effort, coordination behavior, and communication load. These per-rollout results were then aggregated into scenario-level and stack-level summaries. Storing both individual rollout statistics and aggregate results makes it possible to analyze not only average performance, but also variability, outliers, and recurring failure patterns.

The benchmark is intended to evaluate more than goal attainment alone. In the multi-robot setting, reliable navigation depends not only on whether the swarm reaches the goal, but also on how much planning effort is required, how much internal coordination friction arises during traversal, and whether the robots can move safely and efficiently as a group. For this reason, the primary comparison in this chapter is based on six metrics that together capture convergence, computational burden, coordination quality, safety, and motion efficiency.

7.1.4 Illustrative Successful Benchmark Snapshots

Before presenting the quantitative benchmark results, it is useful to visualize representative successful runs of the two evaluated stacks. Figures 7.1 and 7.2 show one intermediate snapshot for each stack during successful traversal in the benchmark environment. The purpose of these images is not to establish a quantitative comparison, but to give visual intuition for the difference between coordinator-guided shared-route execution and self-planned independent-route execution under the same shared-sensing assumptions.

In Figure 7.1, CG-Swarm is shown at an intermediate stage of a successful rollout. In the left panel, the six agents can be seen progressing through the environment while following the shared route. In the right panel, the tunnel-like path structure planned by the coordinator is shown explicitly, illustrating the common navigation reference used by the swarm during traversal.

In Figure 7.2, SP-Swarm is shown at a comparable stage of a successful rollout. Here, the agents do not follow a single shared path, but instead maintain their own individually planned trajectories. The color of each agent corresponds to the color of its path in the figure, making the independent-route organization visible.

In both figures, red obstacles denote obstacles that have already been sensed by the swarm, whereas grey obstacles indicate parts of the environment that remain unsensed at that stage of the rollout. Additional rollout sequences for both evaluated stacks, showing approximately one-third, one-half, and two-thirds progress through successful runs, are provided in Appendix A.3.

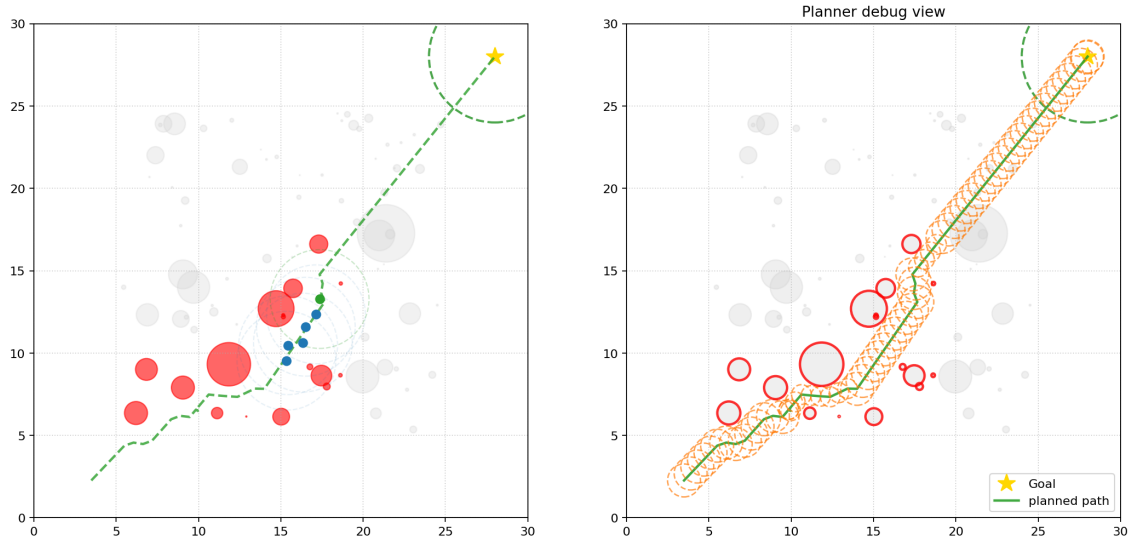

Figure 7.1: Representative successful benchmark snapshot for CG-Swarm at an intermediate stage of traversal. The figure illustrates coordinator-guided shared-route execution using the formation-aware route, tunnel-flocking controller, and motion primitive layer.

7.2 Evaluation Metrics

The swarm benchmark is intended to evaluate more than goal attainment alone. Since swarm navigation requires both individual and collective performance, the primary metrics were selected to reflect convergence, computational burden, coordination quality, safety, and motion efficiency.

7.2.1 Convergence

The primary outcome metric is the *success rate*, defined as the fraction of rollouts in which all agents reached the goal region within the allotted simulation horizon. This is the most direct measure of full-swarm convergence and is therefore the main benchmark metric used to compare the two evaluated stacks.

7.2.2 Computational Burden

To evaluate computational cost, two quantities are considered: *total planning effort* and *wall-clock runtime*. Total planning effort measures the cumulative amount of planning work performed during a rollout and therefore reflects how much replanning and route maintenance each stack requires. Wall-clock runtime complements this by indicating the practical execution cost of the full simulation. Together, these two metrics distinguish stacks that merely succeed from stacks that do so efficiently.

7.2.3 Coordination and Safety

Because swarm navigation requires multiple robots to move through the same environment without obstructing one another, the benchmark also evaluates the burden of local coordination. This is captured through *traffic occasions*, which indicate how often local motion conflicts or yielding situations arise during traversal.

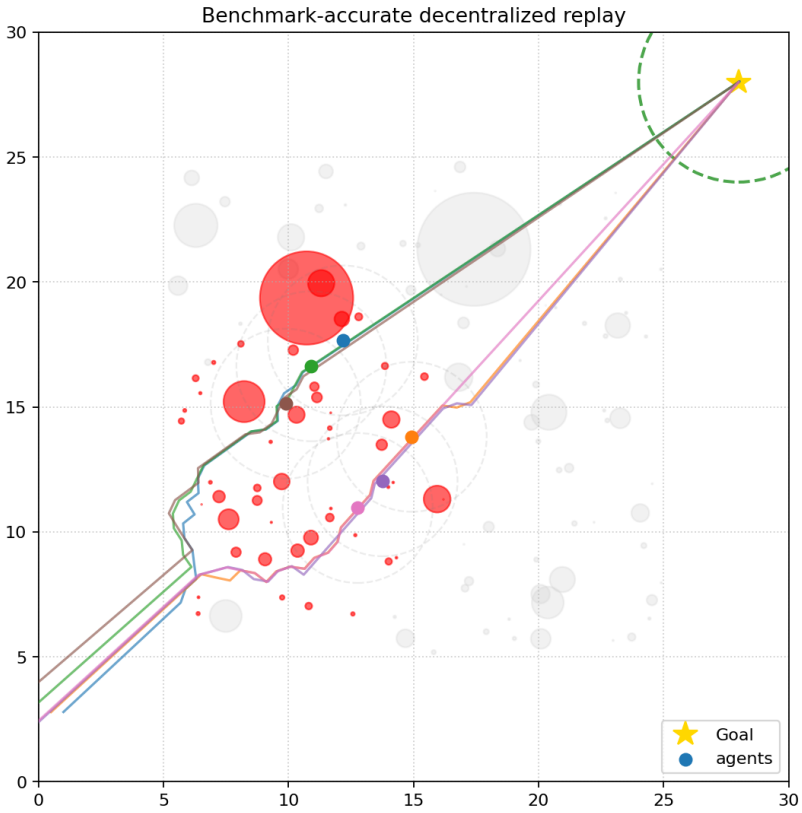

Figure 7.2: Representative successful benchmark snapshot for SP-Swarm at an intermediate stage of traversal. The figure illustrates self-planned independent-route execution with shared obstacle sensing and motion primitive execution.

Internal swarm safety is evaluated through the number of *inter-agent collisions*. In the present benchmark, this metric is particularly important because both evaluated stacks remain highly effective at avoiding static obstacles. As a result, the main observed safety difference lies not in obstacle avoidance itself, but in the ability to regulate internal swarm motion under clutter.

7.2.4 Motion Efficiency

Finally, the benchmark records the *total path length* traveled during each rollout. This provides a measure of execution efficiency at the swarm level. Longer paths may indicate detours, oscillatory progression, or inefficient local interaction, whereas shorter paths indicate more direct and stable collective motion toward the goal.

Additional logged quantities, such as failure categories, message counts, and virtual-obstacle statistics, are used later in the chapter as supporting diagnostics to help interpret the primary metric differences between the evaluated stacks.

7.3 Aggregate Benchmark Results

Figure 7.3 presents the aggregate benchmark comparison between CG-Swarm and SP-Swarm using the six primary comparison metrics: success rate, total planning effort, traffic occasions, wall-clock runtime, inter-agent collisions, and total path length.

Figure 7.3: Aggregate benchmark comparison between CG-Swarm and SP-Swarm over all scenarios, using the six primary evaluation metrics: success rate, total planning effort, traffic occasions, wall-clock runtime, inter-agent collisions, and total path length.

At the aggregate level, CG-Swarm outperformed SP-Swarm across all six metrics. Its success rate was higher, while its planning effort, traffic burden, runtime, inter-agent collision rate, and total path length were all lower. This indicates that the proposed coordinator-guided stack not only achieved more reliable collective arrival, but did so with less planning overhead, fewer local conflicts, and more efficient execution overall.

The strongest differences appear in total planning effort, wall-clock runtime, and inter-agent collisions. These quantities show that the main advantage of CG-Swarm is not merely a modest improvement in arrival probability, but a broader reduction in the instability and coordination burden associated with swarm traversal through cluttered terrain. In contrast, SP-Swarm remained capable of partial and often substantial goal progress, but required much more replanning and exhibited substantially more internal swarm interference.

The total path length results provide an additional efficiency perspective. Although the difference in path length is smaller than the differences in planning effort or runtime, CG-Swarm still produced shorter executed trajectories on average. Thus, SP-Swarm was not trading higher planning cost for shorter swarm motion. Instead, it was both more computationally expensive and less efficient in executed travel distance.

7.4 Scenario-Wise Performance Analysis

To understand how these differences evolve with environmental difficulty, Table 7.2 reports the same six metrics separately for each scenario. This scenario-wise view is important because aggregate values alone can obscure whether a stack degrades gradually or sharply as clutter increases.

Table 7.2: Scenario-wise comparison using the six primary benchmark metrics.

Scen.	Stack	Success	Plan effort	Traffic	Wall time [s]	I-A coll.	Path length [m]
1	CG-Swarm	0.99	836.64	3.09	23.72	0.00	240.79
1	SP-Swarm	0.88	4195.61	29.81	53.29	3.76	244.53
2	CG-Swarm	0.99	1048.05	2.28	27.38	0.00	242.91
2	SP-Swarm	0.89	5813.10	42.76	79.47	2.55	250.57
3	CG-Swarm	0.99	1120.65	15.22	30.38	0.00	243.54
3	SP-Swarm	0.85	5384.50	32.92	89.10	3.00	259.22
4	CG-Swarm	0.96	2669.96	14.99	46.44	0.00	249.60
4	SP-Swarm	0.80	5123.58	33.72	94.97	1.53	270.95
5	CG-Swarm	0.94	1397.92	21.22	39.76	1.13	252.51
5	SP-Swarm	0.63	7768.24	55.00	152.79	5.39	289.98

The scenario-wise results show that CG-Swarm maintained a clear advantage throughout the full scenario family. In the easier scenarios, both stacks were often able to make substantial progress, but CG-Swarm still achieved higher success rates while requiring far less planning effort and exhibiting far fewer traffic interactions. This indicates that self-planning with shared obstacle information does not eliminate coordination friction, even in relatively open environments.

As clutter increased, the performance gap became more pronounced. CG-Swarm remained highly reliable, with success rates of 0.96 and 0.94 in Scenarios 4 and 5, whereas SP-Swarm dropped to 0.80 and 0.63. The same trend appears in runtime and path length. In the most difficult scenario, SP-Swarm not only failed more often, but also required far more computation and produced substantially longer executed trajectories.

The traffic statistics reinforce this interpretation. In every scenario, SP-Swarm experienced many more traffic occasions than CG-Swarm. The difference is especially large in Scenarios 1, 2, and 5. This suggests that the self-planned swarm spent much more time resolving local conflicts between agents, even when both stacks remained obstacle-safe. CG-Swarm therefore appears to reduce not only global planning burden, but also the local friction associated with keeping multiple robots moving through the same environment.

Inter-agent collision counts reveal a similarly important distinction. CG-Swarm recorded no inter-agent collisions in the first four scenarios and only a small nonzero value in Scenario 5. SP-Swarm, in contrast, produced inter-agent collisions in every scenario. This indicates that the major safety difference between the stacks lies not in obstacle handling, but in the ability to regulate internal swarm motion under clutter.

Finally, the path-length results show that CG-Swarm did not achieve its gains by taking substantially longer or more conservative routes. On the contrary, it consistently produced shorter executed path lengths than SP-Swarm. This strengthens the interpretation that the proposed stack improves both reliability and efficiency, rather than simply trading one for the other.

7.5 Failure Breakdown

Table 7.3 summarizes the observed failure categories. The failure distribution further clarifies the difference between the evaluated stacks. CG-Swarm produced only 13 labeled failures in total over the 500 main benchmark episodes: 10 timeout partial-arrival failures, 2 planner failures, and 1 inter-agent collision. In contrast, SP-Swarm produced 95 labeled

failures over the same number of episodes, consisting of 59 timeout partial-arrival failures and 36 inter-agent collisions.

Table 7.3: Failure breakdown by evaluated stack.

Failure reason	CG-Swarm	SP-Swarm
Timeout_PartialArrival	10	59
PlannerFailure	2	0
InterAgentCollision	1	36

The dominant SP-Swarm failure modes are therefore not terrain collisions, but incomplete collective arrival and internal swarm interference. This suggests that shared sensing alone does not provide sufficient structure for robust collective traversal in cluttered environments. CG-Swarm performs better in this respect because its coordination layer constrains agents to a common route, reducing conflicting motion decisions and thereby lowering traffic-like interference within the swarm.

Another notable observation is that inter-agent collisions were present, whereas obstacle collisions were not. This is consistent with the design of the motion primitive layer: candidate motions are checked against static obstacles and against the current positions of neighboring agents using a safety margin before execution. However, these checks remain essentially instantaneous. They account for where neighboring agents are at the decision moment, but not for how those agents may simultaneously move during execution. As a result, collisions with the environment are successfully avoided, while some collisions between agents can still occur due to unmodeled relative motion.

In principle, this limitation could be reduced by replacing the purely position-based neighbour check with a dynamic collision-avoidance formulation based on velocity obstacles. The Reciprocal Velocity Obstacle (RVO) formulation of van den Berg et al. [28], for example, models multi-agent collision avoidance in velocity space by considering relative motion between agents. Optimal Reciprocal Collision Avoidance (ORCA) extends this idea into an optimal reciprocal collision-avoidance formulation, where each agent takes part of the responsibility for avoiding pairwise collisions [29]. Such methods would likely reduce inter-agent collisions because they reason about predicted relative motion rather than only current neighbour positions. However, they would also make the primitive-selection layer more conservative, especially in dense regions where many candidate motions may be rejected. For that reason, velocity-obstacle methods would address the local collision-avoidance limitation, but would not by themselves resolve the broader traffic and coordination issues observed in SP-Swarm.

7.6 Interpretation of the Main Performance Differences

Taken together, the aggregate, scenario-wise, and failure-breakdown results indicate that CG-Swarm provides three main advantages over SP-Swarm.

First, CG-Swarm achieves more reliable *full-swarm convergence*. The difference between the aggregate success rates of 0.974 and 0.810 is substantial, especially given that the single-agent planner already achieves strong individual reachability in earlier chapters. The key challenge at the swarm level is therefore not only whether one robot can reach the goal, but whether all robots can do so together without fragmentation or prolonged local conflict.

Second, CG-Swarm achieves this with much lower *planning overhead*. SP-Swarm requires several times more planning effort in every scenario. This is expected because each agent maintains and repairs its own route, so replanning effort can be duplicated across the swarm. In contrast, CG-Swarm centralizes route maintenance at the coordinator level and provides a shared traversal reference for the group. The results suggest that shared obstacle information alone is insufficient to prevent repeated or distributed replanning effort when each agent plans independently.

Third, CG-Swarm provides stronger *internal swarm regulation*. Since both stacks exhibit zero obstacle collisions on average, the main safety distinction lies inside the swarm itself. SP-Swarm experiences far more inter-agent collisions and traffic-related events, indicating that the difficulty of swarm navigation in this benchmark lies primarily in mutual coordination rather than in obstacle avoidance alone.

The decisive difference between the two evaluated stacks is therefore not that one stack can avoid terrain obstacles while the other cannot. Both can use sensed obstacle information to avoid static hazards. The difference is that CG-Swarm uses communication and planning to maintain a common traversal structure, whereas SP-Swarm leaves agents to resolve most route-level conflicts independently. This makes the self-planned stack more vulnerable to fragmented progress, repeated replanning, and traffic-like interference.

7.7 Discussion of Communication and Coordination Behavior

The communication statistics should be interpreted in light of the benchmark setup. Routine swarm-state updates were applied directly in the benchmark loop rather than transmitted through explicit low-level state messages. Since this simplification was used for both stacks, it does not affect the fairness of the comparison, but it does mean that the reported message counts reflect only the higher-level communication events that remained explicitly modeled.

Within that abstraction, the total communication load is of similar order for both stacks. This is notable because CG-Swarm communicates both obstacle-related information and explicit path updates, whereas SP-Swarm communicates obstacle-related information only. The similar overall message totals are consistent with the observation that SP-Swarm ends up carrying more obstacle information on average.

The benchmark therefore suggests that the stronger performance of CG-Swarm is not explained simply by a much larger amount of explicitly modeled communication. Rather, the key difference is how the available communication is used. In CG-Swarm, communication maintains a shared traversal structure through route updates and coordinated execution. In SP-Swarm, communication improves local obstacle knowledge, but does not impose a common route-level organization on the swarm.

7.8 Illustrative Extreme Cases

The per-rollout extreme-case summaries provide useful qualitative support for the aggregate metrics. The highest-planning-effort and longest-wall-time cases are dominated by SP-Swarm failures, especially timeout partial-arrival and inter-agent collision cases with very large planning effort values. The most severe self-planned outliers exceed 10^5 in total planning effort and more than 1000 seconds in wall time.

CG-Swarm also exhibits outliers, especially in difficult Scenario 5 rollouts with repeated

rejoin events and elevated communication load. However, these outliers are fewer and less dominant in the aggregate statistics. This asymmetry is important: although CG-Swarm is more reliable overall, its rare failures still reveal meaningful limitations of the current implementation.

Two representative extreme cases help illustrate the difference in failure behavior between the stacks. In CG-Swarm, one observed failure mode occurs when rejoin behavior is triggered but the rejoin path interacts unfavorably with a nearby obstacle. In such a case, the affected agent can become trapped near the obstacle instead of successfully returning to the shared traversal structure. Once this occurs, the swarm may continue to wait on or repeatedly react to the stranded agent, resulting in a pathological rollout that remains effectively stuck for the rest of the episode. An example of this behavior is shown in Appendix A.4.1.

In SP-Swarm, a different type of extreme case is observed. When the agents encounter a local minimum or similarly constrained region, some agents may discover a viable way around it, while others fail to do so and instead continue exploring inconsistent alternatives, partially reverting along previously attempted routes or oscillating between competing path choices. This leads to fragmented swarm behavior in which part of the team progresses toward the goal while other agents remain trapped or only make limited progress. Such cases are consistent with the partial-arrival failures and elevated planning effort observed in the benchmark. A representative example is shown in Appendix A.4.2.

These examples connect the aggregate statistics to concrete rollout behavior. The CG-Swarm outlier illustrates that even a stack with strong overall performance can fail when structural recovery breaks down. The SP-Swarm outlier, by contrast, illustrates how independently maintained trajectories can lead to uneven local recovery, prolonged replanning, and incomplete collective arrival.

7.9 Preliminary Scaling Observations

To obtain a first indication of how both stacks behave beyond the six-agent benchmark used in the main evaluation, an exploratory scaling experiment was conducted for swarms of 10, 15, and 20 agents. In contrast to the main benchmark, the purpose of this experiment was not to establish a statistically robust scaling study, but rather to probe how the two architectures behave when the swarm size is increased under the same scenario family.

The results presented here are based on five episodes per scenario, stack, and swarm size. This is sufficient to provide a more informative exploratory picture, but it is still not large enough to support strong benchmark-level conclusions about scaling behavior. The results should therefore be interpreted as preliminary trends only, useful for identifying early patterns and motivating future larger-scale evaluation.

Figure 7.4: Exploratory scaling comparison for swarm sizes of 10, 15, and 20 agents. Bars show the mean over the five scenario classes for the six primary evaluation metrics.

Figure 7.4 suggests a clear qualitative trend. As the swarm size increases, CG-Swarm remains comparatively stable across the reported metrics, while SP-Swarm degrades much more strongly. This difference is especially visible in success rate, total planning effort, traffic occasions, wall-clock runtime, and inter-agent collisions.

For CG-Swarm, success remains high across all three tested swarm sizes, and the corresponding increases in planning effort, traffic burden, and runtime remain relatively moderate. SP-Swarm, in contrast, shows a substantial decline in success rate as the number of agents increases, together with sharp increases in planning effort, local traffic conflict, runtime, and inter-agent collision counts. These observations are consistent with the main six-agent benchmark, where SP-Swarm already exhibited substantially more replanning burden and internal swarm interference than CG-Swarm.

Among the reported quantities, the most striking differences appear in planning effort and inter-agent collisions. As the swarm grows, SP-Swarm appears to incur increasingly heavy planning overhead while also experiencing much more severe internal coordination breakdown. The total path length also increases more strongly for SP-Swarm, suggesting that the additional planning and coordination burden does not translate into more efficient traversal.

These observations should, however, be interpreted carefully. Although the present experiment provides a useful early indication that CG-Swarm may scale more favorably than self-planned swarm navigation in the tested environments, the number of runs is still too limited to support strong claims about general scaling performance. A more extensive scaling benchmark would be needed before drawing robust conclusions about how either stack behaves as swarm size increases more broadly.

The per-scenario scaling results are provided in Appendix A.5, where the exploratory trends in Figure 7.4 can be inspected in more detail.

7.10 Chapter Summary

This chapter evaluated the complete Coordinator-Guided Swarm stack, CG-Swarm, against the Self-Planned Swarm baseline, SP-Swarm. The comparison was designed to test whether applying the improved single-agent I-RAPF planner independently to all agents is sufficient for swarm navigation, or whether collective traversal benefits from coordinator-guided shared-route management.

Across the evaluated scenarios, CG-Swarm achieved a consistent performance advantage over SP-Swarm. This advantage was reflected not only in higher full-swarm convergence reliability, but also in lower planning effort, reduced traffic interactions, shorter wall-clock runtime, fewer inter-agent collisions, and shorter executed path lengths. The results therefore indicate that the benefit of CG-Swarm is not limited to goal-reaching probability, but extends to the overall stability and efficiency of collective traversal.

The failure analysis showed that the dominant SP-Swarm failure modes were incomplete collective arrival and internal swarm interference, rather than terrain collisions. This supports the interpretation that the main difficulty in the tested swarm setting is not only obstacle avoidance, but also organizing how multiple agents use the available free space. Running I-RAPF independently on all agents gives each rover a strong individual planner, but does not by itself impose a shared traversal structure on the group.

The communication analysis further showed that the performance difference is not explained simply by a much larger modeled communication volume. Rather, the key distinction is how communication is used. In CG-Swarm, obstacle information and route updates support a common formation-aware traversal structure. In SP-Swarm, obstacle information improves individual route planning, but route decisions remain independent.

The qualitative extreme cases and preliminary scaling observations reinforce this interpretation. CG-Swarm still exhibits rare failure modes, particularly related to rejoin behaviour after route updates, but these failures are less frequent and less dominant than the fragmented progress, repeated replanning, and inter-agent conflicts observed in SP-Swarm. The exploratory scaling results suggest that the value of shared route organization may increase with swarm size, although a more extensive scaling benchmark would be needed before making strong general claims.

The next chapter complements this simulation-based swarm benchmark with preliminary real-rover validation of I-RAPF on the Lunar Zebro rover platform. That validation focuses on the improved single-agent planner rather than the full CG-Swarm architecture, and therefore provides a first hardware step toward the broader swarm system developed in this thesis.

Chapter 8

Real-Rover Validation of I-RAPF

The previous chapter evaluated the complete Coordinator-Guided Swarm stack, CG-Swarm, in simulation and compared it against the Self-Planned Swarm baseline, SP-Swarm. Those results assessed the proposed swarm architecture under controlled and repeatable benchmark conditions. However, the full swarm benchmark remained simulation-based. To complement this evaluation, this chapter investigates whether the improved single-agent planner developed earlier in the thesis can be integrated and tested on the physical Lunar Zebro rover platform.

The purpose of this chapter is therefore narrower than the purpose of the swarm benchmark. It does not validate the complete CG-Swarm architecture on multiple physical rovers. Instead, it focuses on the single-rover planning component: Improved RAPF (I-RAPF). The experiments compare I-RAPF against the existing RAPF implementation available on the rover and evaluate their behaviour in an indoor obstacle environment. This provides a first hardware-oriented validation step between the simulation results and future multi-rover deployment.

This distinction is important for interpreting the results. The real-rover experiments are not intended to reproduce the full simulation benchmark, nor to establish statistically comprehensive hardware performance. Instead, they test whether the improved planner can operate inside the physical rover software stack, produce executable goal-directed navigation behaviour, and reveal practical issues that are not visible in simulation. The chapter therefore emphasizes deployment feasibility and experimental lessons rather than large-scale performance claims.

The real-rover tests use a set of obstacle scenarios with increasing clutter and multiple start–goal configurations. Navigation behaviour is recorded using ROS bags and processed offline to evaluate goal-reaching performance, completion time, heading-projected travelled distance, and collision-related indicators. Because localization noise and the practical test setup limit the precision of some measurements, the results are interpreted as preliminary validation rather than as a definitive planner benchmark.

The chapter is structured as follows. Section 8.1 defines the purpose and scope of the real-rover tests. Section 8.2 introduces the compared planner configurations. Section 8.3 describes the rover platform, indoor localization setup, and obstacle representation. Section 8.4 presents the scenario design and run matrix. Section 8.5 explains the test procedure and logging process. Section 8.6 defines the evaluation metrics. Section 8.7 presents the results, Section 8.8 discusses their interpretation and limitations, and Section 8.9 summarizes the chapter.

8.1 Purpose and Scope of the Real-Rover Tests

The real-rover validation has two main purposes. First, it provides a physical comparison between RAPF and I-RAPF. Second, it identifies practical deployment issues that are not visible in the simulation-only benchmark, such as localization disturbances, communication or software failures, and differences between planned and executed motion.

The comparison is deliberately limited in scope. The real-rover campaign does not attempt to reproduce the full simulation benchmark on hardware, nor does it validate the complete swarm architecture. Instead, the experiments are designed as controlled repeated trials on a small number of fixed obstacle scenarios. This keeps the physical test campaign manageable while still allowing both planners to be compared under identical conditions.

The tests are performed on a flat indoor floor using the anchor-based localization system. This setup avoids the additional complexity of sandbox terrain, wheel–leg interaction with sand, and terrain deformation. By removing these effects from the first physical validation, the experiment focuses more directly on planner behaviour and on the practical integration of the planner with the rover software stack.

The research questions addressed in this chapter are:

1. Does the I-RAPF planner reach the goal at least as reliably as RAPF under identical real-rover test conditions?
2. Does the I-RAPF planner reduce navigation-related failures, timeouts, or manual aborts in the tested scenarios?
3. Are differences between the planners consistent across the tested obstacle-complexity levels?
4. What practical deployment issues appear when transferring the planner from simulation to the physical rover?

Because the number of real-rover runs is small, the results are interpreted as preliminary physical validation rather than as a statistically exhaustive benchmark. The simulation chapters provide the broad quantitative evaluation, while the present chapter investigates whether the planner comparison remains meaningful when executed on the real rover platform.

8.2 Compared Planner Configurations

Two planner configurations are compared in the real-rover tests. The first configuration is RAPF, the hardware baseline defined at the start of this chapter. The second configuration is I-RAPF, the improved planner developed in this thesis.

Both planners are tested using the same turn-then-move execution policy. This is an important experimental control. The objective of the real-rover validation is not to compare different execution policies, but to compare planner-level behaviour. Keeping the execution policy fixed prevents improvements or failures from being attributed to changes in low-level motion execution.

Table 8.1: Planner configurations compared in the real-rover validation.

Configuration	Planner	Execution policy
C1	RAPF	Turn-then-move
C2	I-RAPF	Turn-then-move

For each test condition, both planner configurations are evaluated using the same obstacle layout, start–goal configuration, localization setup, rover hardware, and test procedure where possible. This creates a paired comparison: every obstacle layout and start–goal configuration tested with RAPF is also tested with I-RAPF.

8.3 Experimental Platform and Test Environment

The real-rover tests were performed in a roughly rectangular indoor test area of approximately 3 m in the x-direction and 2 m in the y-direction. Obstacles were placed in the central part of this area, approximately satisfying $0.5 < x < 2.5$ m, so that the rover had to navigate through a cluttered middle region before reaching the goal-side part of the map. The start poses were placed near one side of the test area, while the goal poses were placed in the region beyond approximately $x = 2.5$ m.

Figure 8.1: Physical validation setup during a representative run of scenario C, corresponding to the scenario definitions in Table 8.2. The obstacle stands are placed in the central part of the workspace, forcing the rover to navigate through the cluttered region before reaching the goal-side part of the test area.

Figure 8.1 shows the physical validation setup for one representative run of scenario C. The anchor stands define the rectangular working area used for the experiments, while the obstacle stands define the cluttered region that the rover must traverse. The shown setup corresponds to one of the obstacle configurations summarized in Table 8.2.

The physical obstacles consisted of vertical stands with approximately truncated-octahedron-shaped obstacle bodies, composed of square and hexagonal faces. For planning, these physical objects were abstracted as circular obstacles with predefined centre positions and radii, as defined in the YAML scenario files.

8.3.1 Lunar Zebro Rover Setup

The experiments are performed using the Lunar Zebro rover platform. The rover is operated through the ROS 2 software stack used by the Lunar Zebro project. During each run, the planner generates a path toward the commanded goal, and the rover follows this path using the turn-then-move execution policy.

The turn-then-move policy first aligns the rover heading with the direction of the next motion segment and then executes forward motion. This policy reflects the non-holonomic nature of the rover: the rover cannot instantaneously move in an arbitrary planar direction, but must orient itself before translating. Since both planner configurations use the same execution policy, the comparison remains focused on the planner output and replanning behaviour.

Before each run, the rover is manually placed at the selected start pose. The goal pose is then sent using the RViz goal tool. The run continues until the rover reaches the goal-side region, a timeout occurs, unsafe behaviour requires interruption, or a failure in hardware, localization, communication, or software invalidates the run.

8.3.2 Indoor Anchor-Based Localization Setup

The real-rover tests are performed in an indoor flat-floor environment using the anchor-based localization system. The use of an indoor setup is intentional. It reduces uncontrolled variability from terrain interaction and allows the two planner configurations to be compared under more repeatable conditions.

The recorded localization message contains the rover position in centimetres and the rover heading in degrees. The heading convention is such that zero degrees points in the positive x-direction and heading increases anticlockwise. Since the analysis relies on recorded rover trajectories, localization quality directly affects the reliability of the extracted metrics. During post-processing, position jitter was found to be large enough that raw point-to-point trajectory integration overestimated travelled distance. For this reason, the path-length metric used in this chapter is a heading-projected distance estimate rather than a raw integrated trajectory length.

8.3.3 YAML-Based Obstacle Representation

Obstacles are represented through YAML scenario files and provided to the navigation stack using the obstacle detector mock. This means that the planner receives obstacle information from the software stack rather than from physical obstacle perception. The obstacle geometry is therefore known from the selected YAML file, including obstacle positions and radii.Across the tested scenarios, the circular obstacle radii ranged from approximately 0.05 m to 0.25 m. In general, the scenarios with fewer obstacles used larger circular obstacle radii, while the scenarios with more obstacles used smaller radii. This avoided making the denser scenarios unrealistically blocked while still increasing the number of obstacle-avoidance decisions required by the planner.

This setup is useful for a first physical planner validation because it keeps the obstacle layout controlled and repeatable. Offline obstacle-intersection checks were attempted by comparing the recorded rover trajectory against the known YAML obstacle geometry. However, the anchor-based localization signal showed sufficient drift and jitter that the resulting binary collision labels were not reliable: many runs were classified as intersecting obstacle regions even when this was not consistent with the practical interpretation of the run. Therefore, binary collision count is not used as a primary quantitative metric in this chapter.

8.4 Scenario Design and Test Conditions

8.4.1 Obstacle Scenarios

The real-rover validation uses three fixed scenario families ordered by increasing obstacle count. Scenario A contains four obstacles, Scenario B contains six obstacles, and Scenario C contains eight obstacles. This ordering makes the scenario labels follow a clearer progression from the least cluttered to the most cluttered test condition, while keeping the hardware campaign small enough to execute and analyse reliably.

Table 8.2: Obstacle scenarios used in the real-rover validation.

Scenario	Number of obstacles	Interpretation
Scenario A	4	Low-obstacle validation case
Scenario B	6	Medium-obstacle validation case
Scenario C	8	Higher-obstacle validation case

The scenarios are not intended to reproduce the full lunar simulation benchmark. Instead, they provide a compact set of physical test conditions for comparing the two planner implementations on the rover. Across the scenario files, the obstacles were placed in the central navigation region of the test area, approximately between $0.5 < x < 2.5$ m. This left a start-side region before the obstacle field and a goal-side region after the obstacle field.

8.4.2 Obstacle Seeds and Start–Goal Configurations

For each scenario family, two obstacle seeds are used. Each seed corresponds to a fixed YAML obstacle file. For every obstacle seed, three start–goal configurations are tested. The start–goal configurations are denoted $g1$, $g2$, and $g3$ in the run logs and represent one approximately straight route through the test area and two diagonal route variants. This gives:

$$3 \text{ scenarios} \times 2 \text{ seeds} \times 3 \text{ start-goal configurations} = 18 \text{ conditions per planner.} \quad (8.1)$$

Using multiple start–goal configurations is important because the same obstacle layout can be easy or difficult depending on the direction from which it is approached. Each of the resulting conditions is tested with both planner configurations.

The goal poses were issued manually using the RViz goal tool. As a result, the exact commanded goal coordinates were not available in the recorded data with sufficient reliability for a precise goal-distance metric. Instead, success is reconstructed from the recorded trajectory using a goal-zone criterion, as described in Section 8.6.

8.4.3 Run Matrix

The final analysed run matrix consists of three scenario families, two seeds per scenario, three start–goal configurations per seed, and two planner configurations. This gives a total of 36 analysed runs:

$$3 \text{ scenarios} \times 2 \text{ seeds} \times 3 \text{ start-goal configurations} \times 2 \text{ planners} = 36 \text{ runs.} \quad (8.2)$$

Table 8.3: Real-rover validation run matrix.

Component	Value
Planner configurations	RAPF, I-RAPF
Execution policy	Turn-then-move
Environment	Indoor flat-floor test area
Localization	Anchor-based localization
Obstacle representation	YAML-defined mock obstacle detections
Scenario families	Scenario A, Scenario B, Scenario C
Obstacle counts	4, 6, and 8 obstacles
Seeds per scenario	2
Start–goal configurations per seed	3
Analysed runs per planner	18
Total analysed runs	36

Some physical conditions required repeated attempts during the test session. In the final analysed dataset, only the last attempt for each planner–scenario–seed–start-goal condition is retained.

8.5 Test Procedure and Logging

8.5.1 Run Execution Procedure

Each run follows the same procedure. First, the correct obstacle YAML file is selected and loaded by the obstacle detector mock. The rover is then placed manually at the predefined start pose. Before starting the run, the operator verifies the rover status, localization availability, selected planner configuration, execution policy, and ROS bag recording.

After setup, the goal pose corresponding to the selected test condition is published using RViz. The navigation command is then sent to start path following. During the run, the operator observes the rover without intervening unless intervention is required for safety or because the run has become invalid.

A run is stopped when one of the following conditions occurs:

1. the rover reaches the goal-side region;
2. the run exceeds the allowed timeout;
3. unsafe behaviour requires manual interruption;
4. a hardware issue prevents continuation;
5. localization becomes unavailable or unreliable;
6. communication or software failure invalidates the run.

After the run, the ROS bag recording is stopped and the run outcome is entered into the manual run log. The rover is then reset to the start pose before the next run.

8.5.2 ROS Bag Recording and Offline Processing

During each run, the localization and parameter-event topics were recorded in a ROS bag. The recorded localization stream is used to reconstruct the rover trajectory. The localization message contains x and y in centimetres and heading h in degrees. Since the message itself does not contain a timestamp field, the time column used in the analysis is obtained from the ROS bag message timestamp.

The real-rover analysis focuses on quantities that can be reconstructed consistently from the localization trajectory: goal-zone success, completion time, active duration, heading-projected travelled distance, final x position, and maximum reached x position.

Because bag recording was started before the rover began moving, the start of each run is detected from the trajectory as the first sustained movement interval. Completion time is then measured relative to this detected movement start rather than relative to the first recorded bag timestamp.

8.6 Evaluation Metrics

The real-rover validation uses a compact set of metrics selected to capture goal-reaching behaviour and execution efficiency using only the recorded data that was reliable enough for quantitative analysis.

Table 8.4: Evaluation metrics used for the analysed real-rover validation experiments.

Metric	Unit	Source
Goal-zone success	Binary or %	Recorded localization trajectory
Completion time to $x > 2.5$ m	s	ROS bag timestamps and detected movement start
Active duration	s	Detected movement interval
Heading-projected distance estimate	m	Recorded position and heading
Final median x position	m	Final localization window
Maximum reached x position	m	Recorded localization trajectory
Obstacle collision	Diagnostic only	Trajectory and YAML obstacle geometry

Goal-zone success. Because the goal poses were issued manually using the RViz goal tool, the exact commanded goal coordinates were not available in the recorded data. The goals were placed in the goal-side region of the test area, which starts at approximately $x = 2.5$ m. A run is therefore counted as successful when the rover comes to rest in this goal-side region:

$$x_{\text{final}} > 2.5 \text{ m}. \tag{8.3}$$

Here, x_{final} is computed as the median x position over the final recorded pose window. This final-position criterion is used throughout the main results.

Completion time. Completion time is measured from the detected start of rover motion until the first time the trajectory crosses $x > 2.5$ m. This metric is only reported for runs that satisfy the $x > 2.5$ m goal-zone success criterion.

Active duration. Active duration is the duration of the detected movement interval. This differs from the full bag duration because the ROS bag recording was started before the autonomous motion began and sometimes continued briefly after the run was effectively complete.

Heading-projected distance estimate. Raw point-to-point integration of the localization trajectory was found to overestimate travelled distance because the anchor-based localization signal contained position jitter. To reduce the effect of lateral localization drift, the reported distance metric is a heading-projected estimate. For two consecutive recorded positions, the displacement is:

$$\Delta \mathbf{p}_k = \begin{bmatrix} x_{k+1} - x_k \\ y_{k+1} - y_k \end{bmatrix}, \tag{8.4}$$

and the heading direction is:

$$\mathbf{e}_{h,k} = \begin{bmatrix} \cos(h_k) \\ \sin(h_k) \end{bmatrix}, \tag{8.5}$$

where h_k is the recorded rover heading in radians. The forward component is then:

$$d_{\mathrm{fwd},k} = \Delta \mathbf{p}_k^{\mathsf{T}} \mathbf{e}_{h,k}. \tag{8.6}$$

Only positive forward components above a small deadband D_{dead} are accumulated:

$$l_{\mathrm{hp}} = \sum_k \max\left(d_{\mathrm{fwd},k} - D_{\mathrm{dead}}, 0\right). \tag{8.7}$$

The resulting value is therefore a heading-projected executed-distance estimate, not an exact wheel-odometry distance.

Obstacle collision. Offline obstacle-intersection checks were attempted using the recorded trajectory and the known YAML obstacle geometry. However, due to localization drift and the conservative obstacle-radius expansion needed to include rover size, the resulting binary collision labels were not sufficiently reliable for planner comparison. Collision is therefore not used as a primary quantitative metric.

8.7 Results

This section reports the outcome of the real-rover validation. The results are interpreted descriptively rather than as a large statistical benchmark. The main purpose is to compare RAPF and I-RAPF under identical physical test conditions and to identify whether the hardware deployment reveals practical issues not present in simulation.

8.7.1 Goal-Reaching Performance

Since the commanded RViz goals were placed in the goal-side part of the map, a run is counted as successful when the final median rover position lies in the region $x > 2.5$ m. Under this criterion, RAPF reached the goal zone in 12 out of 18 runs, corresponding to a success rate of 66.7%. I-RAPF reached the goal zone in 14 out of 18 runs, corresponding to a success rate of 77.8%. The result suggests that I-RAPF retained a small advantage in goal-reaching reliability on the physical rover. However, the difference is modest and should be interpreted cautiously because the physical campaign is small and the success criterion is based on a reconstructed goal-side region rather than an exact commanded goal coordinate.

8.7.2 Scenario-Level Results

Table 8.5 breaks the results down by scenario after ordering the scenario labels by obstacle count. Scenario A and Scenario B show the clearest improvement in success rate for I-RAPF. In Scenario A, I-RAPF succeeded in all six runs, compared with five out of six for RAPF. In Scenario B, I-RAPF also succeeded in all six runs, compared with five out of six for RAPF. Scenario C, the eight-obstacle case, was the most difficult set of runs for both planners, with both planners succeeding in only two out of six runs.

Table 8.5: Scenario-level real-rover validation results. Success is defined as reaching the goal-side region of the map, where the goals were placed beyond approximately $x > 2.5$ m. Mean time and mean heading-projected distance are computed over successful runs only.

Planner	Scenario	Obstacles	Success	Mean time [s]	Mean travelled distance [m]
RAPF	A	4	5/6 (83.3%)	101.45	4.55
RAPF	B	6	5/6 (83.3%)	77.89	4.16
RAPF	C	8	2/6 (33.3%)	153.28	6.15
I-RAPF	A	4	6/6 (100.0%)	102.87	5.03
I-RAPF	B	6	6/6 (100.0%)	89.44	4.41
I-RAPF	C	8	2/6 (33.3%)	89.92	3.79

Scenario C is notable because both planners had the same success count, but the successful I-RAPF runs were faster and had shorter heading-projected distance estimates than the successful runs of RAPF. This suggests that the most cluttered obstacle layouts remained challenging for both planners, but that I-RAPF did not produce worse hardware behaviour in this difficult subset.

8.7.3 Completion Time and Heading-Projected Distance

The scenario-level values in Table 8.5 show that the successful I-RAPF runs were broadly comparable to the successful RAPF runs in terms of completion time and heading-projected distance. Aggregated over all successful runs, I-RAPF had a slightly lower mean completion time to the goal zone, 95.27 s compared with 100.27 s for RAPF. The heading-projected executed-distance estimate was also similar between planners, with 4.59 m for I-RAPF and 4.66 m for RAPF. This indicates that the main difference in the physical validation is the modest improvement in goal-zone reliability rather than a large reduction in travelled distance.

8.7.4 Collision Checks

Offline collision checks were attempted using the known obstacle geometry from the YAML files. However, the localization trajectory showed enough drift and jitter that the binary collision labels were not reliable for comparison. In particular, the combination of localization uncertainty and a conservative collision threshold based on obstacle radius, rover radius, and safety margin led to many runs being flagged as obstacle intersections. Because these labels could not be trusted as physical collisions, collision count is not used as a main result. This is treated as a limitation of the physical dataset rather than as a planner-performance conclusion.

8.8 Discussion

8.8.1 Comparison with Simulation Results

Comparison with the simulation benchmark. The real-rover results should not be interpreted as a direct quantitative reproduction of the simulation benchmark. The real test field was deliberately much denser than the environments used during simulation-based planner development. In the simulation benchmark, the highest obstacle density corresponded to approximately 160 obstacles in a 400 m^2 environment. Scaled to the roughly 4 m^2 real-rover obstacle region, this corresponds to only about 1.6 obstacles. In contrast, the real-rover scenarios contained 4, 6, and 8 obstacles in approximately the same area. The real experiments therefore tested the planner in a substantially more confined and obstacle-dense setting than the one for which the planner parameters were originally optimized.

This difference helps explain why both planners became stuck more often in the real-rover experiments than in the corresponding simulation results. The RAPF-based planner was optimized for larger environments in which dense or difficult obstacle regions can often be avoided by selecting a route through less constrained terrain. In the real-rover setup, the rover was restricted to a small test field, and the obstacles were placed in the central part of the map. As a result, the planner could not simply route around the difficult region; it had to find a feasible path through it. In several cases, the potential-field formulation therefore treated the obstacle cluster as a blocking region, which reduced the number of feasible paths to the goal-side region.

Despite this difference in scale and obstacle density, the real-rover results still provide three useful observations. First, the I-RAPF implementation transferred to the physical rover without requiring changes to the high-level planning logic, indicating that the improvements introduced in Chapter 3 are not limited to the idealized simulation environment. Second, in the lower- and medium-density real scenarios, I-RAPF achieved successful traversal in all trials, while RAPF failed in one trial in each of these scenarios. This suggests that the added robustness mechanisms remain beneficial when the planner is coupled to real localization, actuation, and obstacle-map updates. Third, in the most cluttered scenario, both planners achieved only two successful trials, showing that the real test field exceeded the difficulty range for which the current parameter set was optimized.

The main conclusion is therefore not that the real-rover experiment reproduces the simulation benchmark quantitatively, but that it supports the qualitative trend observed earlier: the I-RAPF modifications improve robustness in feasible cases, while very compact obstacle clusters can still cause the RAPF formulation to classify the environment as effectively blocked. This identifies a remaining transfer gap between the large-scale simulation scenarios and short-range manoeuvring in highly cluttered physical test fields.

Improved performance in this type of compact, highly cluttered environment would likely require a separate parameter optimization study, similar to the Optuna-based tuning procedure described in Chapter 3. Parameters such as the bacterial step size, obstacle repulsion gain, and goal attraction gain could then be tuned specifically for short-range manoeuvring through dense obstacle fields. However, such a parameter set would not necessarily remain optimal for the larger-scale planning problems considered in the simulation benchmark. The real-rover validation should therefore be interpreted as a stress test of the planner transfer to hardware, rather than as a like-for-like replication of the simulation scenarios.

8.8.2 Deployment Issues Observed on Hardware

The physical test campaign exposed several practical issues that are important for future hardware experiments. The most important issue was the quality of the anchor-based localization signal. The recorded position sometimes showed drift or jitter while the rover was stationary. This affected raw trajectory integration and made binary obstacle-collision reconstruction unreliable. The heading-projected distance metric was introduced specifically to reduce the effect of this localization noise on the travelled-distance estimate.

For this reason, the hardware validation focuses on robust trajectory-level indicators rather than precise geometric reconstruction of every event during the run. Success is evaluated using the goal-side region of the test field, while travelled distance is estimated from the recorded rover motion in a way that is less sensitive to stationary localization jitter. Obstacle collisions are therefore treated as diagnostic observations rather than as the primary quantitative performance metric.

These observations should be interpreted as practical lessons for future rover experiments rather than as direct planner-performance metrics. They show that the planner could be evaluated on hardware, but also that more accurate and stable localization would be needed before drawing strong conclusions about fine-grained quantities such as exact collision timing, small trajectory deviations, or precise path length.

8.8.3 Limitations of the Real-Rover Evaluation

The real-rover validation has several limitations. First, the number of physical runs is small. The analysed dataset contains 36 runs, corresponding to 18 conditions per planner. This is sufficient for preliminary validation but not for a statistically exhaustive reliability analysis.

Second, the experiments are performed indoors on a flat floor. This setup improves repeatability and isolates planner-level effects, but it does not include sandbox terrain, wheel–leg interaction with loose soil, or terrain deformation. The results therefore do not fully represent lunar-regolith traversal.

Third, the obstacles are provided through YAML-based mock detection rather than through full onboard perception. This keeps the obstacle layouts controlled and repeatable, but it means that the experiment validates planner deployment and execution under known software-defined obstacle inputs rather than the complete perception-to-navigation pipeline.

8.9 Chapter Summary

This chapter presented a preliminary real-rover validation of Improved RAPF (I-RAPF) on the Lunar Zebro rover platform. The purpose of the evaluation was to test whether

the improved single-agent planner could be integrated into the physical rover navigation stack and compared against the existing RAPF implementation under controlled indoor obstacle conditions.

The experiments used obstacle scenarios of increasing clutter and multiple start–goal configurations. Runs were recorded using ROS bags and processed offline to evaluate goal-reaching behaviour, completion time, heading-projected travelled distance, and collision-related indicators. Because the physical test setup introduced localization noise and practical logging constraints, the results were interpreted as preliminary deployment evidence rather than as a statistically complete hardware benchmark.

The real-rover tests support the practical feasibility of I-RAPF as a single-rover planning component. They show that the improved planner can be deployed in the rover software stack and used for goal-directed navigation in physical obstacle environments. At the same time, the campaign also revealed important experimental limitations. In particular, anchor-based localization noise affected trajectory reconstruction.

These observations define the boundary of the hardware conclusions. The chapter validates the deployment of the improved single-agent planner, but it does not validate the complete CG-Swarm architecture on multiple physical rovers. Full hardware validation of formation-aware route planning, tunnel-flocking coordination, primitive-based execution, and coordinator-guided route management remains future work.

The next chapter closes the thesis by discussing the results across the single-agent, sensing-based, swarm, and real-rover evaluations. It answers the research question, summarizes the main contributions, identifies limitations, and outlines future research directions.

Chapter 9

Discussion, Future Work, and Conclusion

This thesis investigated how the Robust Artificial Potential Field (RAPF) navigation framework can be extended from single-agent reactive planning toward reliable swarm navigation in cluttered lunar-like environments. The work was motivated by the observation that lunar micro-rover swarms require lightweight and reactive navigation methods, but that classical potential-field methods suffer from known reachability limitations, including local minima, oscillatory behaviour, and deadlock-like failure modes. These limitations become more severe in the swarm setting, where robots must not only avoid terrain hazards but also maintain collective progress, avoid one another, and move through passages that provide sufficient clearance for the group.

The thesis therefore developed the navigation problem in stages. First, the baseline RAPF planner was optimized and structurally improved into Improved RAPF (I-RAPF). Second, I-RAPF was evaluated in a sensing-based navigation loop, where obstacle knowledge is incomplete at the start of the episode and updated during execution. Third, the planner was extended to the swarm setting through formation-aware route planning, tunnel-flocking coordination, and motion primitive execution. Finally, the improved single-agent planner was tested on the physical Lunar Zebro rover platform in a preliminary real-rover validation.

This chapter discusses the main findings of the thesis, answers the research question, summarizes the contributions, identifies the main limitations, and outlines directions for future work.

9.1 Discussion

The results of this thesis show that extending RAPF to swarm navigation is not a single modification, but a layered design problem. Each stage of the thesis introduced a different source of difficulty. In the single-agent full-map setting, the main challenge was improving the robustness and efficiency of the RAPF planner itself. In the sensing-based setting, the challenge became maintaining reliable navigation when obstacle knowledge is acquired incrementally and paths must be repeatedly validated and repaired. In the swarm setting, the dominant challenge shifted again: the main difficulty was no longer only whether a path exists for one rover, but whether multiple rovers can use the available free space coherently and safely.

The development of I-RAPF showed that the baseline RAPF formulation could be

strengthened by combining parameter optimization with structural planner improvements. Hyperparameter tuning improved the operating regime of the planner, but it was not sufficient by itself. Midpoint collision checking addressed a geometric inconsistency in which endpoint-valid candidate steps could still pass through obstacles between sampled waypoints. Partial-trajectory repair addressed the inefficiency of full-trajectory restarts after local-minimum events by preserving valid path prefixes and recomputing only the affected suffix. These changes improved both the physical interpretation and the computational behaviour of the planner.

The sensing-based evaluation then showed that I-RAPF could function as a reactive planning component under partial observability. This is important because the intended navigation problem is not an offline planning problem with complete prior map knowledge. Obstacles are discovered during execution, and a previously valid path may become invalid when new terrain information becomes available. By embedding I-RAPF in a loop of sensing, map updating, path validation, replanning, and execution, the planner was evaluated under conditions closer to practical rover navigation. The results showed that I-RAPF maintained high reliability while reducing planning effort and path variability relative to the tuned RAPF baseline.

The swarm benchmark revealed a further and more fundamental issue. Shared sensing alone is not sufficient for robust collective navigation. The self-planned baseline, SP-Swarm, exchanged obstacle detections between agents and used the improved single-agent planner, but each robot still maintained and repaired its own route independently. This made the baseline a meaningful comparison: it was not deprived of obstacle information, and it used the same underlying single-agent planning improvements. Its failures therefore highlight a swarm-level limitation rather than a simple sensing limitation.

The dominant failure modes of SP-Swarm were incomplete collective arrival and internal swarm interference, rather than terrain collisions. This indicates that the main swarm-level challenge in the benchmark was not merely detecting or avoiding static obstacles. Instead, the difficulty was organizing how multiple agents use the same constrained free space. Agents with similar obstacle knowledge can still choose incompatible local routes, obstruct one another, or make progress at different rates. In such cases, local coordination and primitive-level safety checks must continuously compensate for route-level disagreement.

CG-Swarm improved this behaviour by using communication to maintain a shared formation-aware traversal structure. The coordinator-guided stack did not simply communicate more information; it used the available information differently. Obstacle detections were used to validate, repair, and update a common route, while the formation-aware planner ensured that this route was evaluated with respect to an effective swarm footprint. The tunnel-flocking layer then allowed agents to distribute themselves along this shared route, and the motion primitive layer converted desired velocities into feasible local actions. The benchmark results therefore support the central design claim of the thesis: reliable swarm navigation requires route-level structure in addition to local sensing and local coordination.

The comparison between CG-Swarm and SP-Swarm also clarifies the role of formation-aware planning. A path that is feasible for a single rover is not necessarily feasible for a swarm. If route planning ignores the spatial footprint of the group, the execution layer may be asked to solve a problem that was already created at the planning layer. The formation-aware planner addresses this by evaluating candidate route points with respect to a formation radius. In this way, route feasibility is no longer defined only by whether a reference point can pass through the environment, but by whether the swarm can be guided through the environment with sufficient local clearance.

The motion primitive layer adds another important practical constraint. The tunnel-

flocking layer produces continuous desired velocities, but the rover cannot execute arbitrary planar velocity commands. By selecting from a finite set of executable primitives, the final layer bridges the gap between swarm-level desired motion and non-holonomic rover action constraints. This makes the complete method more realistic than a purely velocity-level swarm controller, while also providing a final hard safety filter before motion is applied.

The real-rover validation provides a complementary, but narrower, form of evidence. It showed that the improved planner can be integrated into the physical rover navigation stack and tested in an indoor obstacle environment. However, these tests should be interpreted carefully. They validate the practical deployment of I-RAPF on a single rover, not the full multi-rover CG-Swarm architecture. The real-rover experiments therefore support the feasibility of transferring the improved planning component toward hardware, while full hardware validation of the complete swarm stack remains future work.

Overall, the thesis shows a progression from planner robustness to system-level swarm organization. The main insight is that improving single-agent RAPF is necessary but not sufficient for reliable swarm navigation. The swarm also requires a shared route representation that accounts for group clearance, a local coordination layer that executes this route without forcing rigid formation behaviour, and an execution layer that respects the actual motion constraints of the rover.

9.2 Answers to the Research Question

The central research question of this thesis was:

> How can the Robust Artificial Potential Field (RAPF) navigation framework be extended to enable reliable swarm navigation in highly cluttered environments with densely distributed lunar rocks and craters?

The results of this thesis indicate that RAPF can be extended toward reliable swarm navigation by combining three levels of improvement: single-agent planner robustness, sensing-based replanning, and formation-aware shared-route execution.

First, the RAPF planner must be made robust enough to serve as a reliable planning component. In this thesis, this was done through the development of I-RAPF. The optimized parameter set improved the baseline planner behaviour, while midpoint collision checking and partial-trajectory repair addressed limitations that could not be solved by tuning alone. This produced a planner that preserved the lightweight and reactive character of RAPF, but improved its reachability, physical validity, and computational efficiency.

Second, the planner must operate inside a closed-loop sensing and replanning process. In cluttered lunar-like environments, the rover cannot assume full prior knowledge of all rocks and craters. The navigation system must therefore update its active obstacle set during execution, validate the current path against newly detected obstacles, and replan when the route becomes invalid. The sensing-based evaluation showed that I-RAPF can be used in this role, maintaining high goal-reaching performance while reducing replanning effort compared with the tuned RAPF baseline.

Third, the planner must be extended from individual route generation to formation-aware swarm route management. Running an improved single-agent planner independently on each robot, even with shared obstacle detections, does not fully solve the swarm navigation problem. The SP-Swarm benchmark showed that agents can share obstacle information and still suffer from fragmented progress, traffic-like interference, inter-agent collisions, and partial collective arrival. This means that the swarm requires more than shared sensing: it requires a common traversal structure.

The proposed extension addresses this by introducing the formation-aware improved RAPF planner, FA-I-RAPF. Instead of evaluating candidate route points only for a single reference position, FA-I-RAPF evaluates whether each candidate can support a specified formation radius. The planner therefore generates not only a shared path P, but also a radius profile R_P that describes the available formation footprint along the route. This changes the meaning of route feasibility from individual collision avoidance to swarm-compatible clearance.

The shared route is then executed by the tunnel-flocking controller and the motion primitive layer. The tunnel-flocking layer converts P and R_P into desired velocities for each agent, combining path tracking, tube correction, neighbour interaction, and local obstacle repulsion. This allows the swarm to follow a common route while still deforming locally. The motion primitive layer then selects executable primitives that approximate the desired velocity while satisfying hard local safety constraints. Together, these layers connect route-level swarm feasibility to executable rover motion:

$$(P, R_P) \rightarrow \mathbf{v}_{i,k}^{\text{des}} \rightarrow u_{i,k}.$$

The answer to the research question is therefore not simply to run RAPF on each robot and exchange obstacle detections. The results show that this is insufficient in cluttered environments because the agents may still make incompatible route choices. Instead, RAPF can be extended to swarm navigation by using it as the basis for formation-aware shared-route management. The planner determines where the swarm can move, the flocking layer determines how individual agents move through that route, and the primitive layer determines which motions can actually be executed safely.

In this sense, the thesis demonstrates that the lightweight reactive nature of RAPF can be preserved while making the method more suitable for swarm navigation. The key extension is to move part of the swarm constraint into the planning layer: the route itself must be planned with the spatial footprint of the swarm in mind. Local coordination remains necessary, but it is no longer expected to compensate for routes that were never suitable for collective traversal.

9.3 Main Contributions

This thesis makes seven main contributions toward reliable potential-field-based navigation for lunar rover swarms.

1. Development of Improved RAPF for robust single-agent navigation

The first contribution is the development of Improved RAPF (I-RAPF), a strengthened version of the baseline RAPF planner. The improvement process combined systematic hyperparameter optimization with structural modifications to the planning algorithm. The optimized parameter set provided a stronger operating regime for the scenario family considered in this thesis, while midpoint collision checking and partial-trajectory repair addressed limitations that could not be solved through tuning alone.

Midpoint collision checking improved the physical validity of generated paths by reducing the possibility that a discrete candidate transition passes through an obstacle between two collision-free endpoints. Partial-trajectory repair reduced the computational cost of local-minimum recovery by preserving valid path prefixes instead of restarting the full trajectory after every artificial-obstacle insertion. Together, these changes produced a planner that retained the lightweight reactive nature of RAPF while improving reliability and planning efficiency.

2. Evaluation of I-RAPF under sensing-based partial observability

The second contribution is the evaluation of I-RAPF in a sensing-based navigation loop. Rather than assuming complete map knowledge, the sensing-based formulation required the rover to build its active obstacle set during execution. The planner was therefore evaluated as part of a closed-loop process involving sensing, path validation, replanning, and motion execution.

This evaluation showed that the improved planner remained effective when obstacle information was incomplete and updated online. It also clarified the role of I-RAPF as more than an offline trajectory generator: the planner can serve as a reactive route-generation and route-repair component in a navigation system where the environment is revealed incrementally.

3. Formulation of a layered swarm navigation architecture

The third contribution is the formulation of a layered swarm navigation architecture for extending I-RAPF from one rover to multiple agents. The architecture separates the swarm navigation problem into route management, local coordination, motion execution, and communication. This separation was introduced because swarm navigation imposes requirements that are not present in the single-agent setting: shared obstacle discovery, route maintenance under partial observability, inter-agent safety, and route compatibility with the spatial footprint of the group.

The resulting architecture provides a structured way to connect planning and execution. Route-level reasoning determines where the swarm should move, local coordination determines how agents distribute themselves while following the route, and motion execution determines which actions can be applied by the rover platform. This layered decomposition makes the full system modular while still allowing the layers to exchange the information required for coordinated traversal.

4. Formation-aware extension of I-RAPF

The fourth contribution is the formation-aware improved RAPF planner, FA-I-RAPF. This planner extends the I-RAPF candidate-selection process by evaluating whether candidate route points can support an effective formation radius. Instead of generating a path only for a single reference point, the planner generates a shared path P together with a formation-radius profile R_P.

This changes the meaning of route feasibility. A route is no longer considered only in terms of individual obstacle clearance, but also in terms of whether the swarm footprint can pass through the available free space. The formation-radius profile allows the planner to prefer wider routes where possible while still permitting compression through narrow regions when necessary. This provides the route-level structure needed by the downstream coordination layer.

5. Integration of tunnel-flocking and motion primitive execution

The fifth contribution is the integration of formation-aware route planning with local tunnel-flocking coordination and primitive-based execution. The tunnel-flocking layer converts the shared path and radius profile into desired velocities for each agent. These velocities combine path following, tube correction, neighbour interaction, and local obstacle repulsion, allowing the swarm to follow a common route while maintaining flexible local motion.

The motion primitive layer then converts these desired velocities into finite executable rover manoeuvres. This is necessary because the rover cannot directly execute arbitrary planar velocity commands. By rolling out candidate primitives, rejecting unsafe motions, and scoring the remaining feasible primitives, the execution layer connects the continuous swarm-level controller to the rover's discrete non-holonomic action constraints. The complete stack therefore links route-level feasibility, local swarm coordination, and executable robot motion.

6. Comparative benchmark of coordinator-guided and self-planned swarm navigation

The sixth contribution is the benchmark comparison between the proposed Coordinator-Guided Swarm stack (CG-Swarm) and the Self-Planned Swarm baseline (SP-Swarm). Both stacks used shared obstacle detections and the same primitive execution layer, but differed in route organization. CG-Swarm maintained a shared formation-aware route through coordinator-guided route management, while SP-Swarm allowed each agent to plan and repair its own route independently.

This comparison isolated the value of shared formation-aware route structure. The results showed that CG-Swarm achieved higher full-swarm success, lower planning effort, fewer traffic-like interactions, fewer inter-agent collisions, and shorter executed paths. The benchmark therefore supports the conclusion that robust swarm navigation requires more than shared sensing. Communication must be used to organize the swarm around a common traversal structure, not only to distribute obstacle information.

7. Preliminary real-rover validation of I-RAPF

A final practical contribution is the preliminary validation of I-RAPF on the physical Lunar Zebro rover platform. These experiments tested the improved planner in an indoor obstacle setup and compared it with the RAPF implementation available on the rover. The real-rover tests showed that I-RAPF could be integrated into the physical navigation stack and used to perform goal-directed navigation on hardware.

This validation should be interpreted as a first hardware step rather than as a full experimental proof of the complete swarm system. The real-rover campaign evaluated the single-rover planner, not the multi-agent CG-Swarm stack. Nevertheless, it provides evidence that the improved planning component is compatible with the physical rover platform and identifies practical deployment issues that should be addressed before larger multi-rover experiments.

9.4 Limitations

The results of this thesis support the proposed extension of RAPF toward formation-aware swarm navigation, but they should be interpreted within the scope of the assumptions and experimental setup used in this work. The main limitations concern the environment model, sensing and communication assumptions, the abstraction of rover dynamics, the scale of the hardware validation, and the remaining failure modes observed in the swarm benchmark.

Simplified lunar terrain representation

The simulated environments are based on lunar-inspired obstacle distributions, but the terrain itself is represented in a simplified two-dimensional form. Rocks and crater-like hazards are modeled as circular obstacles with associated radii, and traversability is evaluated through geometric clearance. This abstraction is useful for isolating the navigation problem and for comparing planner variants under controlled conditions, but it does not capture the full complexity of real lunar terrain.

In practice, rover traversal depends not only on planar obstacle clearance, but also on slope, wheel–terrain interaction, soil properties, illumination, localization uncertainty, and the three-dimensional shape of rocks and crater rims. A route that is collision-free in the two-dimensional benchmark may therefore still be difficult or unsafe for a physical rover if the terrain geometry affects traction, stability, or wheel contact. The results should consequently be interpreted as evidence for navigation-level feasibility in cluttered environments, rather than as a complete terrain-traversability guarantee.

Idealized sensing and obstacle representation

The sensing-based simulations assume that obstacles within sensing range can be detected and incorporated into the active obstacle set in a consistent geometric form. This allows the experiments to focus on the interaction between local sensing, path validation, replanning, and swarm coordination. However, real perception systems introduce additional uncertainty. Obstacles may be partially observed, misclassified, delayed, duplicated, or estimated with inaccurate positions and sizes.

The real-rover experiments already indicated that localization quality can affect the interpretation of recorded trajectories. Similar uncertainty would also affect online obstacle detection and map construction in a fully autonomous deployment. Since the simulated sensing model does not explicitly represent false positives, false negatives, delayed detections, or uncertainty-aware obstacle fusion, the reported reliability should be understood as performance under bounded but relatively clean perception assumptions.

Simplified communication model

The swarm benchmark models communication through a bounded communication radius and explicit exchange of relevant navigation information. This captures the local nature of communication, but does not fully model bandwidth limits, packet loss, communication delay, asynchronous message timing, or degraded connectivity over irregular terrain. In addition, some routine swarm-state information was handled directly in the benchmark loop rather than transmitted through low-level communication messages.

This abstraction was applied consistently to both CG-Swarm and SP-Swarm, so the comparison between the two stacks remains meaningful. However, the absolute communication behaviour of the proposed method should not be interpreted as a complete communication-system evaluation. In a real deployment, delayed or missing obstacle detections and route updates could affect both route validity and local coordination.

Rover dynamics and primitive abstraction

The motion primitive layer improves realism compared with directly applying continuous planar velocity commands, because it restricts execution to a finite set of short manoeuvres. Nevertheless, the primitive model remains an abstraction of the physical rover. The benchmark does not include detailed actuator dynamics, wheel slip, terrain-dependent

speed variation, motor saturation, or low-level control errors. It also assumes that the predicted primitive rollout is a sufficiently accurate representation of the motion that will be executed.

This means that the primitive layer should be interpreted as a navigation-level bridge between desired swarm motion and executable rover actions, rather than as a full dynamic model of the Lunar Zebro platform. More detailed hardware modeling could change the effective feasibility of some primitives, especially in cluttered or uneven terrain.

Limited dynamic collision prediction between agents

The motion primitive layer applies hard safety filtering against obstacles and neighbouring agents before a primitive is selected. This was effective at preventing terrain collisions in the swarm benchmark. However, the neighbour check remains limited because it primarily evaluates neighbouring agents at their current positions, or through short conservative checks, rather than fully predicting the simultaneous future motion of all nearby agents.

This limitation explains why inter-agent collisions can still occur even when obstacle collisions are avoided. Two agents may each select a primitive that appears safe with respect to the current neighbour positions, while their simultaneous execution creates an unsafe relative motion. The issue is especially relevant for SP-Swarm, where agents may approach the same region from different independently planned routes. More predictive collision-avoidance methods would be required to address this limitation fully.

Simulation-heavy validation of the full swarm stack

The full CG-Swarm stack was evaluated in simulation. This was necessary because repeated swarm experiments with multiple physical Lunar Zebro rovers were outside the practical scope of this thesis. Simulation allowed controlled comparison across many randomized scenarios and provided detailed logging of planning effort, traffic events, inter-agent collisions, and failure modes.

However, the strongest evidence for the complete swarm architecture remains simulation-based. The real-rover validation supports the deployability of the improved single-agent planner, but it does not validate the full formation-aware planner, tunnel-flocking layer, motion primitive layer, and coordinator-guided route management on multiple physical robots. The transition from simulated swarm performance to physical multi-rover deployment therefore remains an important open step.

Limited swarm-size scaling analysis

The main swarm benchmark used a fixed swarm size, with exploratory additional results for larger swarms. These results provide early insight into scaling behaviour, but they do not constitute a complete scalability analysis. Larger swarms may introduce qualitatively different effects, including denser inter-agent traffic, more frequent communication constraints, increased route-update complexity, and more difficult formation compression in narrow passages.

The results therefore support the proposed method for the evaluated swarm size and scenario family, but they do not establish general scaling guarantees for significantly larger rover swarms.

Remaining algorithmic failure modes

Although CG-Swarm substantially outperformed SP-Swarm, the benchmark still revealed failure cases. Some failures were associated with rejoin behaviour after route updates, where an agent could struggle to reconnect to the shared traversal structure. Other failures were related to local interactions between primitive execution, nearby obstacles, and inter-agent motion. In the single-agent sensing benchmark, rare timeout failures were associated with excessive accumulation of artificial obstacles, which could create self-induced local minima.

These failure modes show that the proposed architecture improves reliability, but does not eliminate all forms of navigation failure. In particular, artificial-obstacle management, rejoin logic, and dynamic inter-agent avoidance remain important targets for future improvement.

9.5 Future Work

The limitations above suggest several directions for future research. These directions concern both algorithmic improvements and experimental validation. The most important next steps are improving artificial-obstacle management, strengthening rejoin and inter-agent avoidance behaviour, extending the environmental and sensing models, and validating the full swarm stack on hardware.

Artificial-obstacle pruning and memory management

One important direction is the development of a more adaptive artificial-obstacle management strategy. In both RAPF and I-RAPF, artificial obstacles are used to prevent the planner from repeatedly returning to local-minimum regions. This mechanism is effective in most runs, but the sensing-based benchmark showed that excessive artificial-obstacle accumulation can occasionally over-constrain the search space and create self-induced local minima.

Future work should therefore investigate pruning, decay, or confidence-based management of artificial obstacles. For example, artificial obstacles could gradually lose influence after repeated successful replanning around them, be removed when they no longer affect the active path, or be assigned a finite lifetime based on the number of replanning events. Another option is to distinguish between artificial obstacles caused by true geometric dead-ends and those caused by temporary sensing or execution conditions. Such mechanisms could preserve the benefit of local-minima memory while reducing the risk of long-term over-constraining behaviour.

More robust rejoin behaviour after route updates

The swarm benchmark showed that rejoin behaviour remains an important source of rare CG-Swarm failures. When the shared route is repaired or replaced, agents that are far from the new path must reconnect to the common traversal structure. In the current implementation, this rejoin process can fail when the agent is close to obstacles, when the old and new routes diverge strongly, or when local primitive constraints prevent smooth reconnection.

Future work should therefore develop a more explicit rejoin planner. Instead of relying mainly on local correction toward the updated route, the system could compute short rejoin trajectories that are checked against obstacles, neighbours, and tube constraints.

The coordinator could also assign temporary rejoin targets along the new route to avoid multiple agents attempting to merge at the same point. This would make route repair more robust and reduce the chance that a single delayed agent causes partial swarm arrival.

Predictive inter-agent collision avoidance

The current primitive layer checks candidate actions against obstacles and neighbouring agents, but it does not fully account for the simultaneous future motion of other agents. This limitation explains why some inter-agent collisions can still occur even when candidate motions are filtered before execution. Future work should incorporate more predictive multi-agent collision avoidance.

A natural direction is to integrate velocity-obstacle or reciprocal collision-avoidance ideas into the primitive-selection layer. Rather than checking only current neighbour positions, each candidate primitive could be evaluated against predicted relative motion over the primitive horizon. This would allow the controller to reject primitives that are likely to produce future pairwise conflicts, even if the current positions still satisfy the safety margin. Such a method would be especially useful in dense passages, where agents must coordinate not only where they are, but also how they are moving relative to one another.

This extension should be designed carefully. More predictive collision avoidance may reduce inter-agent collisions, but it may also make the primitive layer more conservative and increase waiting or deadlock behaviour in narrow regions. Future work should therefore evaluate the trade-off between safety, progress, and traffic efficiency.

Uncertainty-aware sensing and mapping

The sensing model used in this thesis assumes relatively clean obstacle detection within a bounded range. Real rover perception is more uncertain. Obstacle positions, radii, and classifications may be noisy, delayed, duplicated, or incomplete. Future work should therefore extend the sensing-based navigation loop to include uncertainty-aware obstacle mapping.

This could include probabilistic obstacle representations, confidence scores for detected hazards, and data-association logic for merging repeated detections of the same object. Path validation could then account for uncertainty by inflating obstacle margins when confidence is low or by triggering active sensing behaviours in ambiguous regions. Such extensions would make the planner more suitable for deployment with real perception pipelines rather than idealized obstacle inputs.

More realistic terrain and rover interaction models

The environmental model can also be extended beyond circular obstacle clearance. Lunar terrain contains slopes, irregular rocks, soft soil, shadowed regions, and surface roughness that may affect rover stability and traction. Future benchmarks should therefore include richer traversability models in which the cost of a route depends not only on planar obstacle avoidance, but also on terrain difficulty.

For the Lunar Zebro platform, this could be combined with a more detailed motion model that includes wheel slip, terrain-dependent speed variation, actuator limits, and localization uncertainty. The motion primitive set could then be evaluated not only for geometric collision safety, but also for expected traversal reliability. This would move the primitive layer from a kinematic execution abstraction toward a more physically grounded rover-navigation model.

Communication delay, loss, and asynchronous coordination

The benchmark comparison between CG-Swarm and SP-Swarm showed that the way communication is used matters more than raw message volume. However, the communication model remains simplified. Future work should evaluate the proposed architecture under more realistic communication constraints, including message delay, packet loss, limited bandwidth, and temporary disconnection.

This is especially important for coordinator-guided route management. If route updates arrive late or are missed by some agents, the swarm may temporarily operate with inconsistent path information. Future versions of the architecture should therefore include explicit handling of stale route versions, acknowledgement of route updates, and fallback behaviours for agents that lose contact with the coordinator. This would make the shared-route concept more robust under realistic multi-robot communication conditions.

Larger and more diverse swarm-scaling studies

The exploratory scaling results suggest that swarm size affects planning effort, communication, local traffic, and the difficulty of maintaining coordinated motion. However, a full scaling study was outside the scope of this thesis. Future work should therefore evaluate CG-Swarm and related variants over a wider range of swarm sizes, obstacle densities, communication radii, and formation-radius sets.

Such a study should examine not only average success rate, but also how failure modes change with swarm size. For example, larger swarms may require multiple route corridors, dynamic subgrouping, or explicit traffic scheduling in narrow passages. This would help determine when a single shared route remains sufficient and when the swarm requires a more distributed or hierarchical route-management strategy.

Hardware validation of the full swarm architecture

The most important experimental next step is hardware validation of the complete swarm stack. The real-rover tests in this thesis provide preliminary validation of I-RAPF on a single physical Lunar Zebro rover, but the full CG-Swarm architecture has not yet been tested with multiple physical rovers.

Future hardware experiments should therefore begin with small multi-rover tests in controlled indoor environments. A suitable progression would be to first validate shared obstacle detection and route broadcasting, then test tube-flocking along a fixed shared route, and finally evaluate full formation-aware replanning with multiple agents. Such staged validation would make it possible to isolate failure sources before moving to larger and more cluttered experiments.

In the longer term, the system should be tested with onboard sensing, imperfect localization, and communication constraints that more closely resemble planetary deployment conditions. This would provide the strongest evidence for whether formation-aware RAPF can serve as a practical navigation framework for lunar micro-rover swarms.

Alternative coordinator and route-management strategies

Finally, future work could investigate alternatives to the single-coordinator route-management structure used in this thesis. The coordinator-guided stack performed well in simulation, but a single coordinator can become a bottleneck or point of failure in larger or less connected swarms. More robust architectures could use dynamic coordinator selection, multiple local coordinators, or distributed agreement on route updates.

A hybrid approach may also be useful. For example, the swarm could follow a shared formation-aware route in cluttered regions, but allow temporary subgrouping or self-planning in open regions. This would preserve the benefits of shared traversal structure where it is needed most, while reducing coordination overhead where the environment allows more independent motion. Such extensions would further explore the balance between centralized route structure and decentralized swarm flexibility.

9.6 Final Conclusion

This thesis set out to investigate how the Robust Artificial Potential Field navigation framework can be extended from single-agent planning toward reliable swarm navigation in cluttered lunar-like environments. The results show that this extension requires more than applying RAPF independently to multiple agents. Reliable swarm navigation depends on improving the robustness of the planner, embedding it in a sensing-based replanning loop, and introducing route-level reasoning about the spatial footprint of the group.

The development of I-RAPF improved the single-agent planner by addressing both physical path validity and inefficient recovery behaviour. The sensing-based evaluation then showed that the improved planner can remain effective when obstacle knowledge is incomplete and acquired during execution. Building on this foundation, the formation-aware swarm architecture extended RAPF from individual route planning to shared route management. By generating a common path and radius profile, the coordinator-guided stack provided the swarm with a traversal structure that local coordination alone could not supply.

The benchmark comparison between CG-Swarm and SP-Swarm showed that shared obstacle sensing is not sufficient by itself. Even when agents exchange detections and use the same improved planner, independent route planning can still lead to fragmented progress, repeated replanning, traffic-like interference, and partial collective arrival. CG-Swarm reduced these effects by using communication to maintain a shared formation-aware route and by executing that route through tunnel-flocking and primitive-based motion selection.

The main conclusion is therefore that the lightweight and reactive character of RAPF can be preserved while extending it toward swarm navigation, provided that the swarm constraint is introduced at the route-planning level. The planner must reason not only about where a single rover can pass, but also about where the swarm can move as a coordinated group. This formation-aware shared-route principle proved effective in simulation and was supported by preliminary real-rover validation of the improved single-agent planner. Full multi-rover hardware validation remains an important next step, but the results indicate that formation-aware RAPF is a promising direction for reliable navigation of lunar micro-rover swarms.

Chapter A

Appendix

A.1 Sensing-Based Benchmark Distribution Plots

The main results chapter reports the sensing-based benchmark metrics as mean ± one standard deviation over 500 episodes per scenario. This provides a compact numerical comparison between Tuned RAPF and I-RAPF. To complement those tables, this appendix presents boxplots for the same metrics. The boxplots show the median, interquartile range, and whisker range of the per-episode distributions.

For readability, outlier markers are omitted from the boxplots. The whiskers extend to 1.5 times the interquartile range. The omitted outlier markers are not removed from the underlying data; they are only hidden in the visualization to make the central distribution easier to inspect.

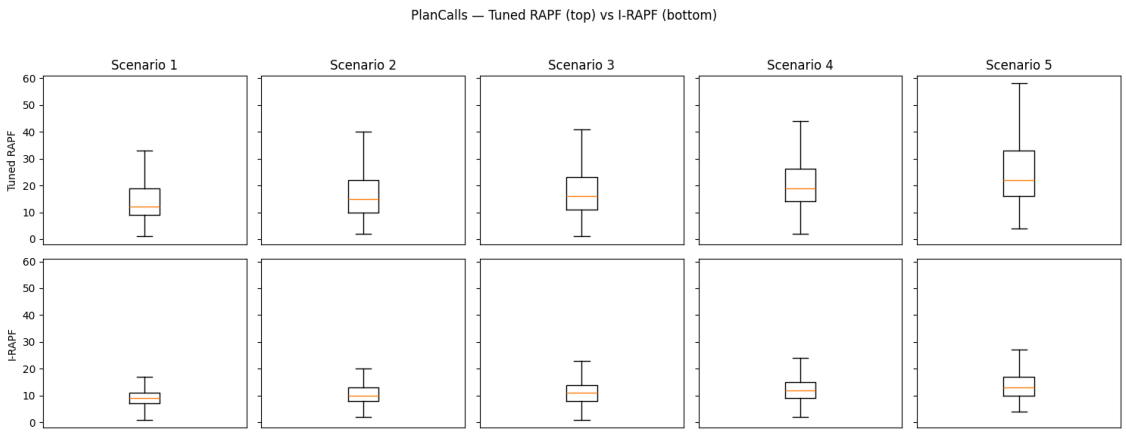

Figure A.1: Distribution of plan calls per scenario for Tuned RAPF and I-RAPF. Each distribution contains 500 episodes. Boxes indicate the interquartile range, the central line indicates the median, and whiskers extend to 1.5 times the interquartile range. Outlier markers are omitted for readability.

Figure A.1 shows that I-RAPF reduces not only the mean number of plan calls, but also the median and interquartile range across all scenarios. The difference becomes more pronounced in the more complex scenarios, where Tuned RAPF shows a wider spread and higher upper whiskers.

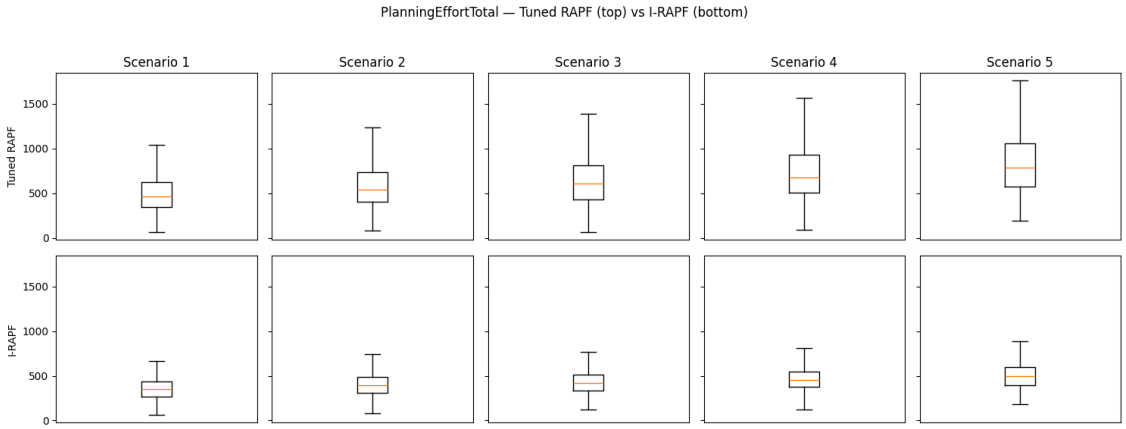

Figure A.2: Distribution of total planning effort per scenario for Tuned RAPF and I-RAPF. Each distribution contains 500 episodes. Boxes indicate the interquartile range, the central line indicates the median, and whiskers extend to 1.5 times the interquartile range. Outlier markers are omitted for readability.

Figure A.2 confirms the same trend for cumulative planning effort. I-RAPF has lower central values and narrower interquartile ranges in each scenario. This supports the interpretation that the algorithmic improvements reduce the computational burden of replanning under partial observability.

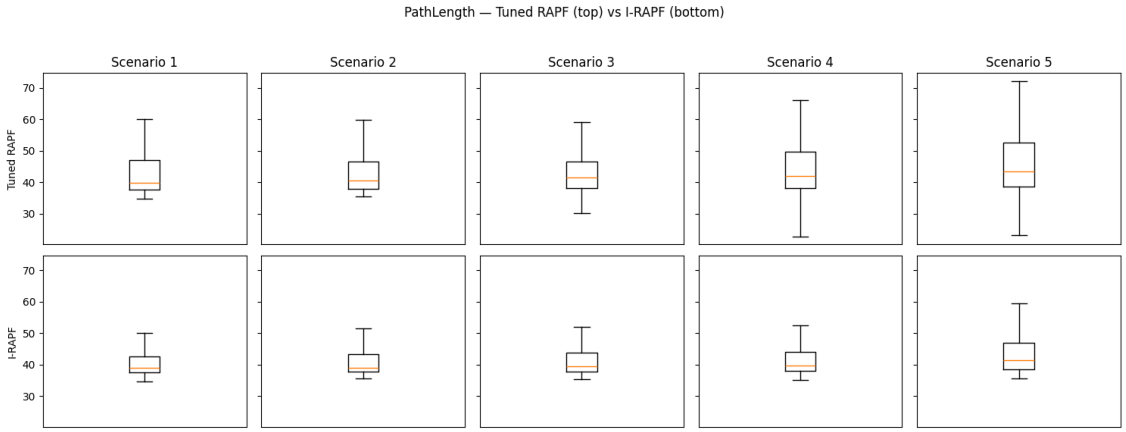

Figure A.3: Distribution of executed path length per scenario for Tuned RAPF and I-RAPF. Each distribution contains 500 episodes. Boxes indicate the interquartile range, the central line indicates the median, and whiskers extend to 1.5 times the interquartile range. Outlier markers are omitted for readability.

Figure A.3 shows that the path-length distributions are closer together than the planning-efficiency distributions. Nevertheless, I-RAPF generally has lower medians and smaller interquartile ranges. This indicates that the reduction in planning effort does not come at the cost of longer executed trajectories.

A.2 Effect of Virtual Obstacle Reset

This appendix illustrates the effect of periodically resetting artificial obstacles during replanning. In the original configuration, the agent becomes trapped due to the accumulation of virtual obstacles, resulting in a timeout. When introducing a reset mechanism every fifth replanning iteration, the agent is able to escape the constrained region and find a valid path to the goal. In the baseline configuration, shown in figure 4.3 without obstacle resetting, the agent timed out after creating 96 artificial obstacles. When the reset mechanism is introduced, only 28 artificial obstacles are needed to solve the same environment, significantly reducing unnecessary space blocking and computational overhead.

Figures A.4, A.5 and A.6 show a comparison between the original behavior and the modified approach.

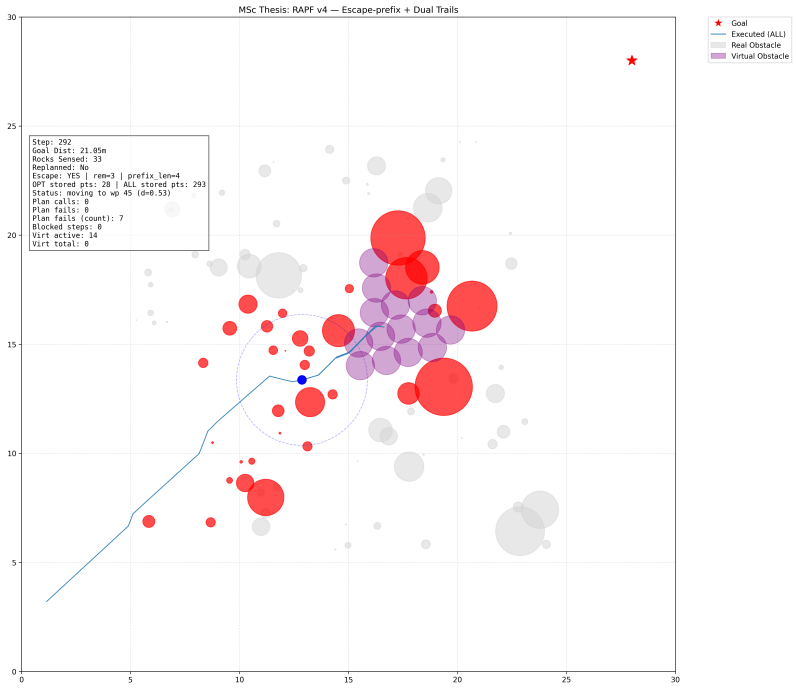

Figure A.4: Path blocked by accumulated artificial obstacles, leading to a timeout configuration

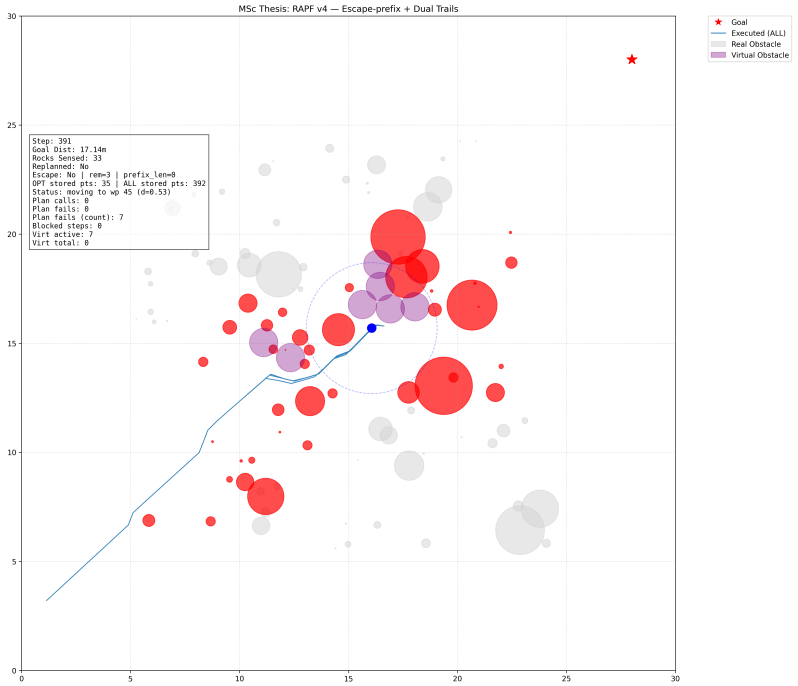

Figure A.5: Artificial obstacles reset during replanning, reopening previously constrained regions

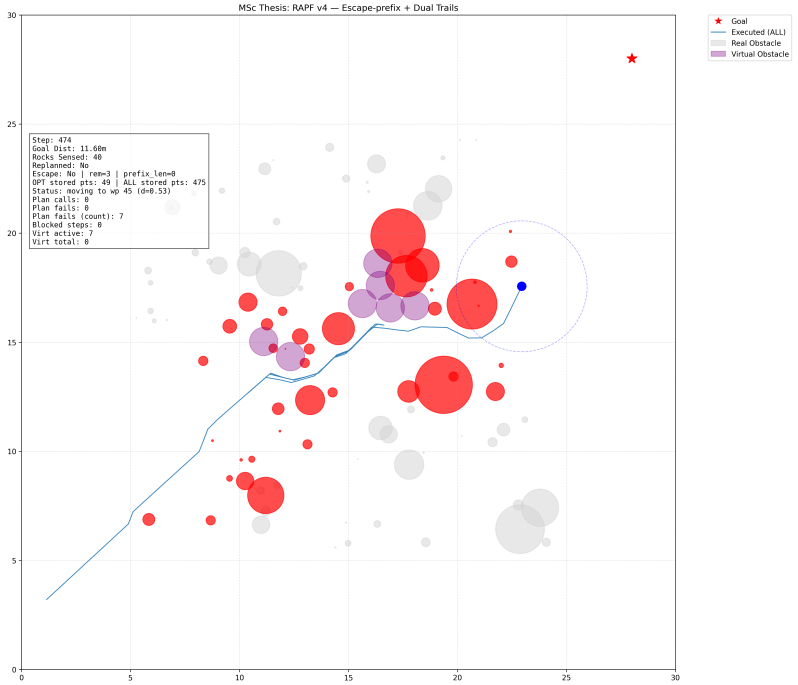

Figure A.6: Recovered trajectory through a newly accessible gap after obstacle reset

-RAPF produced shorter average executed path lengths than Tuned RAPF in every scenario. The standard deviation of the path length was also lower for I-RAPF across the scenario set. The difference becomes most visible in the denser scenarios, where Tuned

RAPF shows both longer mean paths and larger variation. These values indicate that the improved planner does not achieve its higher success rate by producing longer or more conservative trajectories. Instead, under the tested sensing conditions, I-RAPF reaches the goal more reliably while also producing shorter and more consistent executed paths.

A.3 Additional Qualitative Rollout Visualizations

This appendix provides additional qualitative rollout visualizations for the two benchmarked swarm-navigation methods. For each method, three snapshots are shown at approximately one-third, one-half, and two-thirds progress through a successful rollout. These figures complement the representative benchmark snapshots shown in Chapter 11 by illustrating the temporal progression of the swarm through the environment.

In all figures, red obstacles denote obstacles that have already been sensed by the swarm, whereas grey obstacles indicate parts of the environment that remain unsensed at that stage of the rollout.

A.3.1 Coordinator–Follower Successful Rollout

The following figures show a successful coordinator–follower rollout at three stages of traversal. The six agents move through the environment while following the shared route maintained by the coordinator. The accompanying planned-path view shows the tunnel-like path structure used as the common traversal reference during execution.

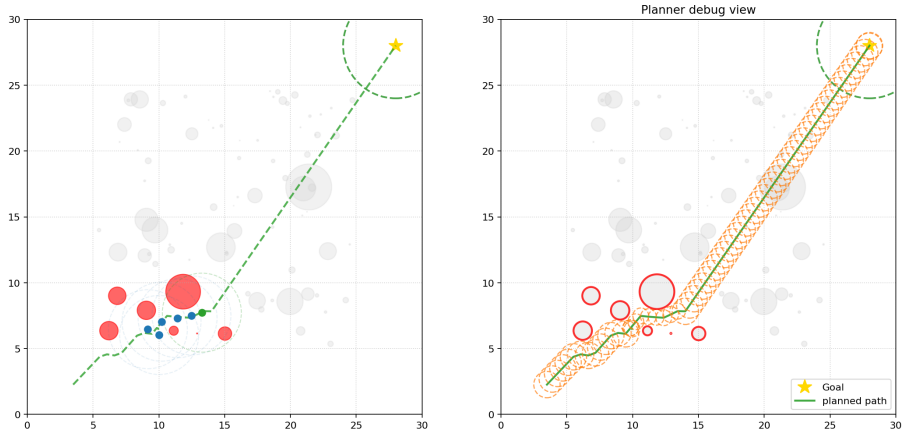

Figure A.7: Coordinator–follower rollout at approximately one-third progress.

A.3.2 Decentralized Successful Rollout

The following figures show a successful decentralized rollout at three stages of traversal. In contrast to the coordinator–follower method, the agents do not follow a single shared path, but instead maintain their own individually planned trajectories. The color of each agent corresponds to the color of its associated path.

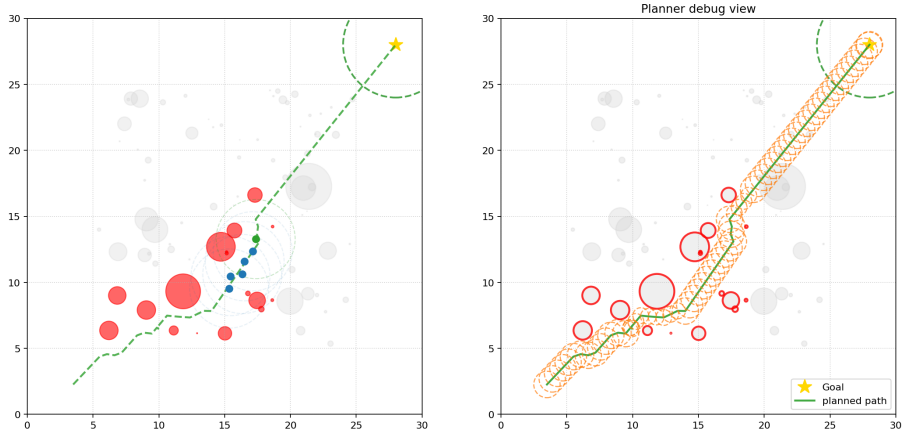

Figure A.8: Coordinator–follower rollout at approximately one-half progress.

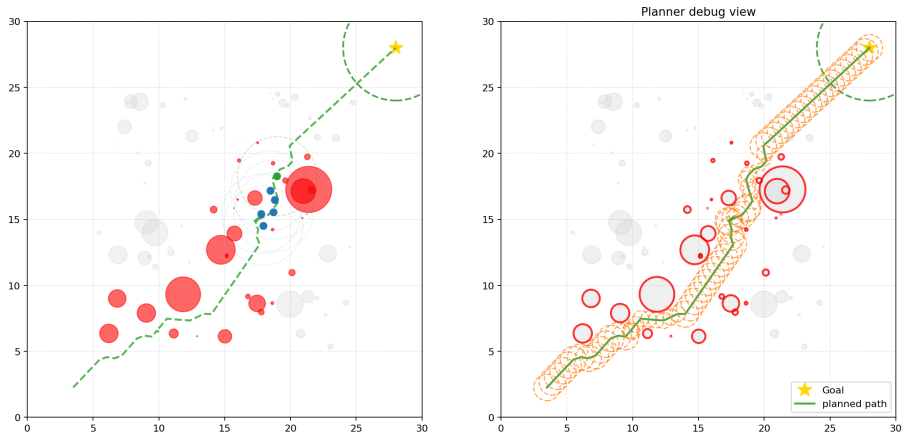

Figure A.9: Coordinator–follower rollout at approximately two-thirds progress.

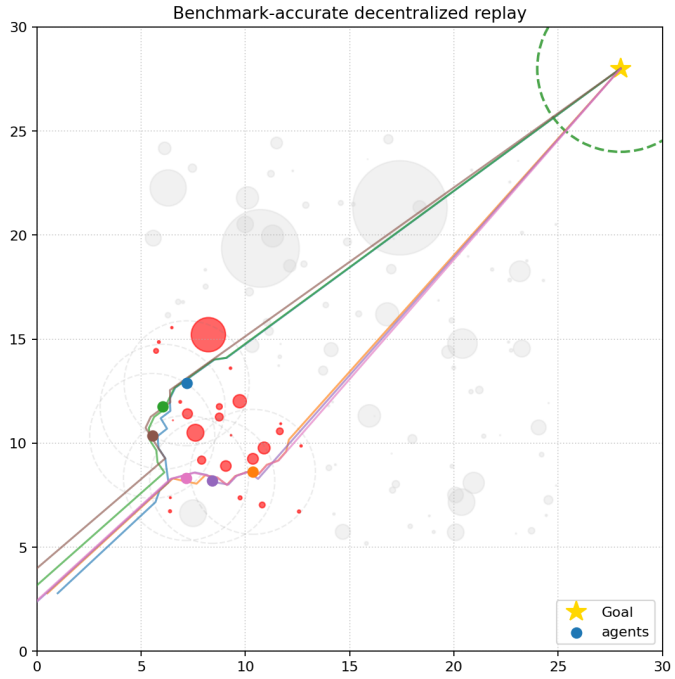

Figure A.10: Decentralized rollout at approximately one-third progress.

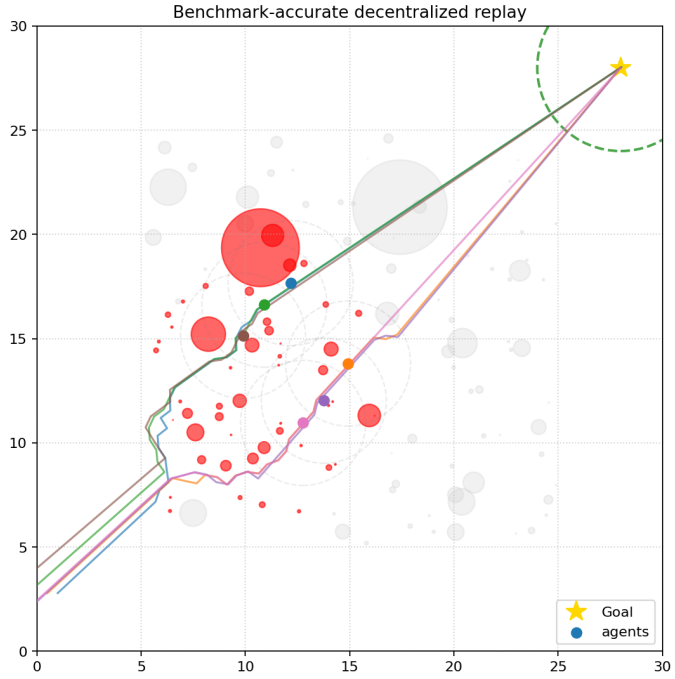

Figure A.11: Decentralized rollout at approximately one-half progress.

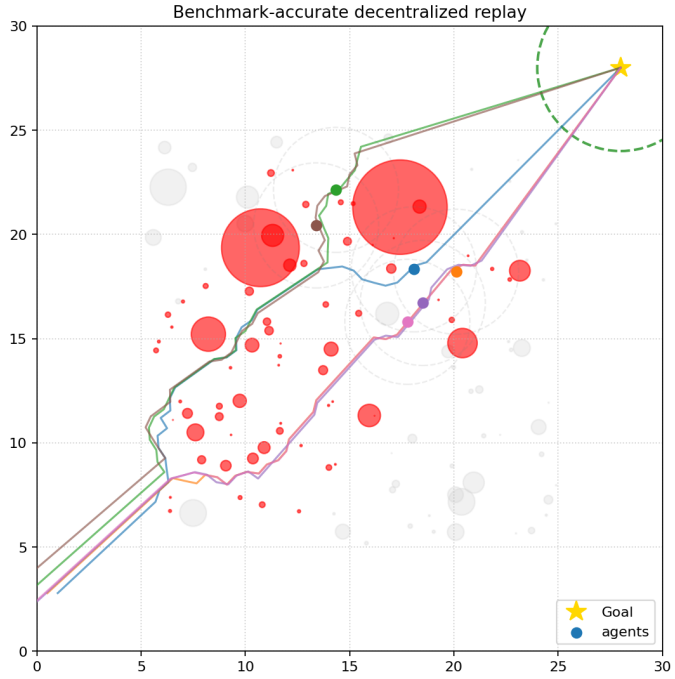

Figure A.12: Decentralized rollout at approximately two-thirds progress.

A.4 Additional Failure-Case Rollout Visualizations

This appendix provides additional qualitative visualizations of representative failure cases observed in the benchmark. The figures are intended to complement the discussion in Chapter 11 by showing how characteristic extreme cases develop in time.

In all figures, red obstacles denote obstacles that have already been sensed by the swarm, whereas grey obstacles indicate parts of the environment that remain unsensed at that stage of the rollout.

A.4.1 Coordinator–Follower Rejoin Failure Case

Figure A.13 shows a representative extreme case of the coordinator–follower method. In this rollout, rejoin behavior is triggered, but the affected agent interacts unfavorably with a nearby obstacle and fails to recover back to the shared traversal structure. The resulting state persists for the remainder of the rollout, producing an extreme case of prolonged stagnation.

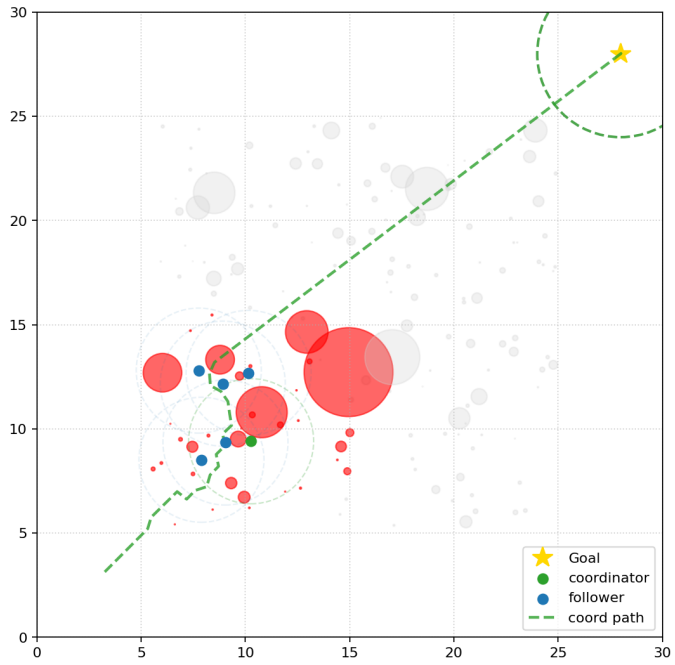

Figure A.13: Coordinator–follower failure case at approximately one-half progress through the rollout.

A.4.2 Decentralized Partial-Arrival / Local-Minima Failure Case

The following Figures illustrates a characteristic extreme case of the decentralized method. When the agents encounter a locally constrained region, some agents discover a viable route around it, while others repeatedly revert toward previously attempted directions. The result is fragmented swarm behavior in which part of the team progresses toward the goal, whereas other agents continue replanning with limited net progress.

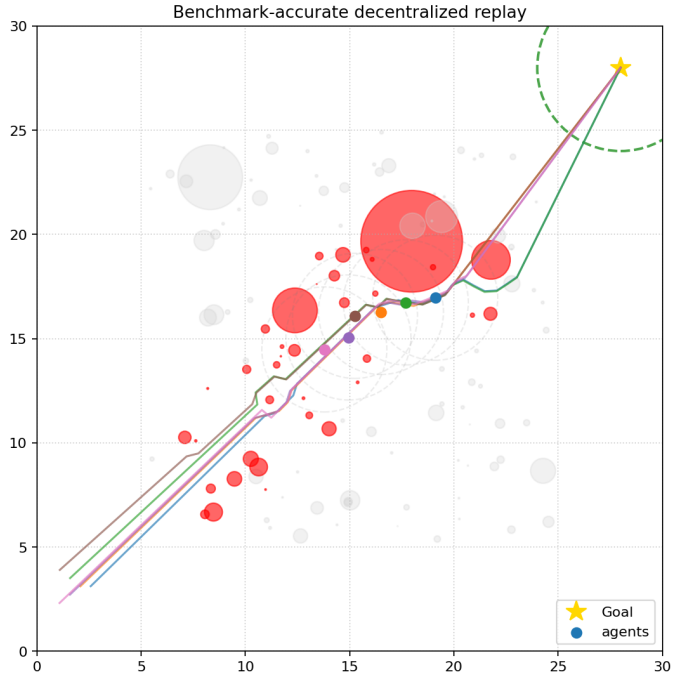

Figure A.14: Decentralized failure case at approximately one-third progress.

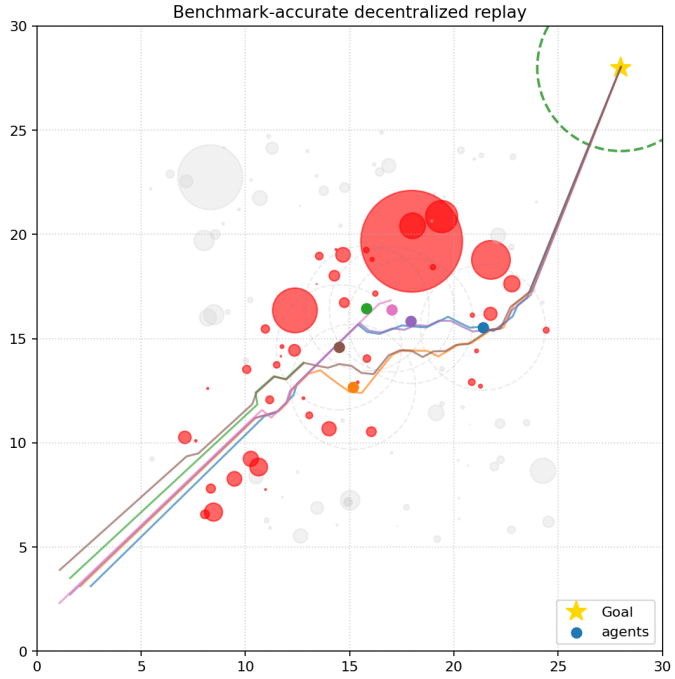

Figure A.15: Decentralized failure case at approximately one-half progress.

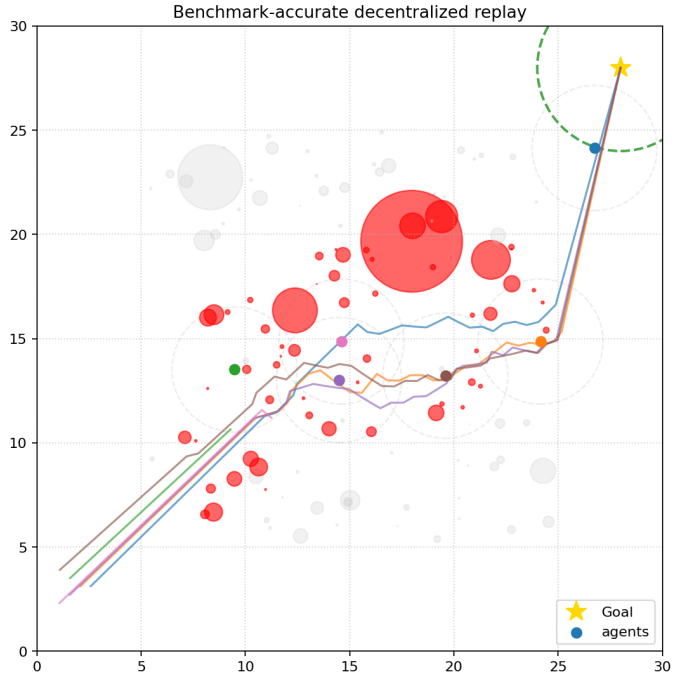

Figure A.16: Decentralized failure case at approximately two-thirds progress.

A.5 Exploratory Swarm-Scaling Results

This appendix provides the per-scenario results of the exploratory swarm-scaling experiment discussed in Chapter 11. The experiment evaluated swarm sizes of 10, 15, and 20 agents for both the coordinator–follower and decentralized methods. For each configuration, the reported values are averages over five episodes per scenario.

The purpose of this experiment was to obtain a first indication of how both methods behave when the swarm size increases beyond the six-agent benchmark used in the main evaluation. Although the use of five episodes per scenario provides a more informative picture than the earlier pilot runs, the experiment should still be interpreted as exploratory rather than as a definitive scaling benchmark. The results are therefore most useful for identifying early trends and possible scaling concerns.

Table A.1 reports the per-scenario averages for the five diagnostic metrics emphasized in the scaling analysis: total planning effort, traffic occasions, wall-clock runtime, inter-agent collisions, and total path length.

Table A.1: Per-scenario results of the exploratory swarm-scaling experiment for swarm sizes of 10, 15, and 20 agents. Values are averaged over five episodes per scenario.

Agents	Scen.	Method	Plan effort	Traffic	Wall time [s]	I-A coll.	Path length
10	1	Coord.-Follow.	876.4	11.0	30.33	0.0	432.70
10	1	Decent.	6401.2	354.2	85.54	103.0	522.21
10	2	Coord.-Follow.	1171.4	13.2	35.47	0.0	449.13
10	2	Decent.	5916.6	422.6	106.04	76.0	497.14
10	3	Coord.-Follow.	942.4	14.0	33.49	0.6	424.23
10	3	Decent.	4659.4	108.2	73.02	14.4	424.89
10	4	Coord.-Follow.	1109.4	11.2	38.34	0.0	418.09
10	4	Decent.	7133.6	569.2	111.73	179.2	516.96
10	5	Coord.-Follow.	1522.2	14.6	44.98	0.0	434.62
10	5	Decent.	6112.2	662.0	159.07	123.8	567.25
15	1	Coord.-Follow.	1082.4	17.0	50.66	0.0	695.26
15	1	Decent.	10577.0	790.0	151.12	222.2	905.62
15	2	Coord.-Follow.	951.8	64.8	57.56	2.2	699.32
15	2	Decent.	9818.2	608.4	165.51	184.6	807.16
15	3	Coord.-Follow.	858.4	14.2	53.91	0.0	587.22
15	3	Decent.	10327.8	1196.6	176.66	443.2	821.60
15	4	Coord.-Follow.	1273.4	16.8	66.78	0.0	687.30
15	4	Decent.	20606.2	1297.4	369.81	360.8	889.61
15	5	Coord.-Follow.	1358.6	20.6	82.31	2.4	752.68
15	5	Decent.	37041.4	758.0	693.15	105.2	910.04
20	1	Coord.-Follow.	1164.4	33.2	75.76	0.0	1000.44
20	1	Decent.	11786.2	2214.6	251.03	437.6	1271.86
20	2	Coord.-Follow.	1150.2	23.2	76.01	0.0	973.67
20	2	Decent.	32361.0	952.4	399.04	300.0	1211.16
20	3	Coord.-Follow.	915.6	19.8	79.35	0.0	964.31
20	3	Decent.	26630.6	1997.6	363.68	539.0	1089.00
20	4	Coord.-Follow.	1477.6	33.4	95.38	0.0	989.04
20	4	Decent.	11875.0	1778.8	296.04	561.4	1277.24
20	5	Coord.-Follow.	1254.0	23.6	101.21	0.0	1002.77
20	5	Decent.	10281.8	1581.4	347.12	365.4	1358.15

Across the reported scenarios, the coordinator–follower method remains comparatively stable as swarm size increases. Its planning effort, traffic burden, runtime, and path length

all increase with swarm size, as expected, but these increases remain moderate and inter-agent collisions remain close to zero in nearly all reported cases. The decentralized method, by contrast, shows much sharper growth in planning effort, traffic occasions, runtime, and inter-agent collisions, particularly for 15 and 20 agents.

These appendix results are consistent with the aggregate scaling figure shown in Chapter 11. At the same time, they also highlight that the severity of the degradation depends on the scenario. In particular, the more cluttered scenarios tend to amplify the gap between methods more strongly than the easier cases. Because this experiment remains exploratory, these observations should be treated as preliminary indications rather than as benchmark-level conclusions.

Bibliography

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] S. Koenig and M. Likhachev, "D* lite," in *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 476–483, 2002.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[4] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[5] F. Matoui, B. Boussaid, and M. N. Abdelkrim, "Local minimum solution for the potential field method in multiple robot motion planning task," in *16th International Conference on Sciences and Techniques of Automatic Control & Computer Engineering (STA)*, pp. 452–457, IEEE, 2015.

[6] D. Puriyanto *et al.*, "Implementation of improved artificial potential field path planning algorithm in differential drive mobile robot," in *Proceedings of the 2022 International Conference on Control, Automation and Robotics (ICCAR)*, pp. 1–6, 2022.

[7] D. Wu, Y. Li, and J. Hu, "Improved artificial potential field method for robot path planning," *Procedia Computer Science*, vol. 163, pp. 529–535, 2019.

[8] H. Li, Y. Wang, and M. Liu, "Combining artificial potential fields with bug algorithms for improved mobile robot navigation," in *2023 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1102–1107, IEEE, 2023.

[9] T. Manteaux, D. Rodríguez-Martínez, and R. T. Rajan, "RAPF: Efficient path planning for lunar microrovers," in *2024 International Conference on Space Robotics (iSpaRo)*, pp. 56–63, IEEE, 2024.

[10] G. Wagner and H. Choset, "Subdimensional expansion for multirobot path planning," *Artificial Intelligence*, vol. 219, pp. 1–24, 2015.

[11] X. Bao, "Advanced apf for real-time swarm path planning with leader-follower logic," *Simulation in Webots for Lunar Terrain*, 2023.

[12] R. Adderson and co authors, "Decentralized time-varying formation with dynamic leader selection based on goal orientation," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[13] M. I. I. Abdi, M. U. Khan, A. Güneş, and D. Mishra, "Escaping local minima in path planning using a robust bacterial foraging algorithm," *Applied Sciences*, vol. 10, no. 21, p. 7905, 2020. Open Access under CC BY license.

[14] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, 2006.

[15] S. Qu, C. Chen, and D. Dong, "Behavior-based control of swarm robots with improved potential field," in *Proceedings of the 33rd Chinese Control Conference*, (Nanjing, China), pp. 8462–8467, July 2014. July 28–30, 2014.

[16] J. V. Gómez, A. Lumbier, S. Garrido, and L. Moreno, "Planning robot formations with fast marching square including uncertainty conditions," *Robotics and Autonomous Systems*, vol. 61, no. 2, pp. 137–152, 2013.

[17] H. Lurz, T. Recker, and A. Raatz, "Spline-based path planning and reconfiguration for rigid multi-robot formations," *Procedia CIRP*, vol. 106, pp. 174–179, 2022.

[18] R. Sharma, T. Weiss, and M. Kallmann, "Formation-aware planning and navigation with corridor shortest path maps," *Computer Graphics Forum*, vol. 43, no. 1, p. e14995, 2024.

[19] W. Liu, J. Hu, H. Zhang, M. Y. Wang, and Z. Xiong, "A novel graph-based motion planner of multi-mobile robot systems with formation and obstacle constraints," *arXiv preprint arXiv:2210.03340*, 2022.

[20] B. Peng, L. Zhang, and R. Xiong, "Smooth path planning with subharmonic artificial potential field," in *International Conference on Advanced Robotics and Mechatronics (ICARM)*, IEEE, 2024.

[21] M. Diab, M. Mohammadkarimi, and R. T. Rajan, "Artificial potential field-based path planning for cluttered environments," in *IEEE Aerospace Conference*, IEEE, 2023.

[22] M. G. Park and M. C. Lee, "A new technique to escape local minimum in artificial potential field based path planning," *Journal of Mechanical Science and Technology*, vol. 17, no. 12, pp. 1876–1885, 2003. Available at: `https://www.researchgate.net/publication/225362938`.

[23] O. Hachour, "Modified artificial potential field for mobile robot path planning in complex environments," in *2017 5th International Conference on Electrical Engineering – Boumerdes (ICEE-B)*, pp. 1–6, IEEE, 2017.

[24] W. Yang, P. Wu, X. Zhou, H. Lv, X. Liu, G. Zhang, Z. Hou, and W. Wang, "Improved artificial potential field and dynamic window method for amphibious robot fish path planning," *Applied Sciences*, vol. 11, no. 5, p. 2114, 2021.

[25] Q. Lv, G. Hao, Z. Huang, B. Li, D. Fu, H. Zhao, W. Chen, and S. Chen, "Localized path planning for mobile robots based on a dynamic-window and improved artificial potential field approach," *International Journal of Advanced Robotic Systems*, vol. 10, no. 1, p. 25, 2024.

[26] L. Yang, Y. Che, J. Guo, and X. Huang, "A local multi-robot cooperative formation control," in *Proceedings of the 33rd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, (Nanjing, China), May 2018. May 18–20, 2018.

[27] P. Mao, R. Fu, and Q. Quan, "Optimal virtual tube planning and control for swarm robotics," *International Journal of Advanced Robotic Systems*, pp. 1–24, 2024. Also available as arXiv:2304.11407.

[28] J. van den Berg, M. C. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1928–1935, 2008.

[29] J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, pp. 3–19, Springer, 2011.