ASSESSMENT COVERSHEET

ENG40008/10 Engineering Technology Project B

Semester 2, 2025

Project Title: Multi-Base Dynamic Role Tracking of a Fast Target with Swarm Robots

Group ID: XG-1 – G55

Supervisor's Name: Dr. Xiaohua (Jamie) Ge

Student Details:

Student ID	Given Name*	Family Name*	Major*	Signature	Date
103600160	Gabriel	Barrasso	Robotics and Mechatronics	Gabriel Barrasso	3/11/25
103597673	Gabriel	Twigg-Ho	Robotics and Mechatronics	Gabriel Twigg-Ho	3/11/25

DECLARATION

We declare that (the first four boxes must be completed for the assignment/report to be accepted):

☑ This assignment/report contains no material that has previously been submitted for assessment at this or any other university. It contains no materials previously published or written by another person except where due reference has been made. Where a report has been professionally edited, the student must include separate acknowledgement that such "editing has been and the editing has addressed only the style and/or grammar of the report and not its substantive content."

☑ This is an original piece of work and no part has been completed by any other student than those signed below.

☑ We have read and understood the avoiding plagiarism guidelines at http://www.swinburne.edu.au/ltas/plagiarism/students.htm and no part of this work has been copied or paraphrased **from any other source** except where this has been clearly acknowledged in the body of the assignment and included in the reference list.

☑ We have retained a copy of this assignment/report in the event of it becoming lost or damaged.

☑ (optional) We agree to a copy of the assignment/report being retained as an exemplar for future students (subject to identifying details being removed).

Date of Submission: 3rd November 2025

Section	Topic	Group/Individual	Contributors	Words
-	ABSTRACT	Group	G.TH/G.B	134
1.	INTRODUCTION	Group	G.TH/G.B	612
2.	LITERATURE REVIEW	Group	G.TH/G.B	2816
3.	DESIGN REQUIREMENTS & METHODOLOGY	Group	G.TH/G.B	1032
3.1	SYSTEM DESIGN AND DEVELOPMENT	Group	G.TH/G.B	1012
3.2	IMPLEMENTATION	Group	G.TH/G.B	2658
3.3	RESTULTS	Group	G.TH/G.B	1813
3.4	DISCUSSION	Group	G.TH/G.B	1552
3.5	CONCLUSION	Group	G.TH/G.B	879

Name	Total Words Contributed (Individual sections)	Total Words Contributed (Group Sections)	Total Words Contributed
G.TH	~	12508	12508
G.B	~	12508	12508

1 ACKNOWLEDGMENTS

We would like to express our sincere gratitude our supervisor, Dr. Xiaohua (Jamie) Ge. His expertise guided this project from its initial conception to its final form. Dr. Ge's insightful feedback and encouragement during our challenging early phases, particularly in navigating the complexities of swarm robotics literature, was invaluable. His responsiveness and transparency was also very valuable in the support of this project. We are grateful for the support and commitment Jamie has provided throughout the project and to our development as student researchers.

2 ABSTRACT

This thesis presents a modular, 2D swarm robotics simulator designed to investigate deployment-centric strategies for the persistent tracking of a target moving at twice the speed of any individual agent. The research methodology builds upon a Particle Swarm Optimization (PSO) inspired framework evolving from a baseline implementation that yielded a modest 14.6% line-of-sight (LOS) persistence to a highly effective hybrid control system. Through systematic optimization and iterative design we developed a novel state-dependent control system that delegates tasks to specialized interception, pursuit and search algorithms based on the tactical context. This adaptive architecture achieved a final LOS persistence of 94.41% a more than six-fold improvement over the baseline. The results demonstrate that our deployment centric, behaviourally adaptive approach provides a robust and resource efficient solution to the challenging fast target pursuit problem.

3 CONTENTS

4 LIST OF FIGURES

5 LIST OF TABLES

6 LIST OF SYMBOLS AND ABBREVIATIONS

Abbreviation	Meaning
LOS	Line-Of-Sight
PSO	Particle Swarm Optimization
UAV	Unmanned Aerial Vehicle
SAR	Search and Rescue
GIF	Get-In-Front (Behaviour)
ISR	Inverse Square Repulsion (Behaviour)

7 INTRODUCTION

Background and Context

The domain of autonomous systems has seen advancements, with applications spanning from industrial automation to planetary exploration. A cornerstone challenge within this field, particularly for surveillance, security and Search and Rescue operations, is the persistent tracking of dynamic targets [1]. The problem is significantly amplified when the target possesses superior mobility such as a faster vehicle being pursued by a swarm of slower ground or aerial drones. In such scenarios no single agent can maintain contact necessitating a coordinated collective strategy to ensure persistent observation. This thesis addresses this complex problem by investigating how a decentralized swarm of autonomous agents can effectively track a target moving at twice the speed of any individual agent.

Motivation and Real-World Relevance

The motivation for this research is rooted in real world applications where timely and persistent tracking is critical. Examples include law enforcement pursuing a suspect vehicle through a suburban environment, environmental agencies monitoring fast moving wildlife or disaster response teams tracking a drifting vessel in a maritime search and rescue context [1] [2]. In these situations, the ability to maintain an unbroken line-of-sight (LOS) is paramount for situational awareness and successful mission outcomes. Traditional approaches often rely on a single, high capability asset, which represents a single point of failure. Swarm robotics, conversely offers a paradigm of decentralized control distributing decision making and sensing across multiple, simpler agents [3]. This approach promises enhanced robustness, scalability, and flexibility, making it an ideal framework for addressing the fast-target pursuit problem.

Research Foundation Aims and Hypothesis

This project began with the development of a modular 2D swarm tracking simulator in Python. The initial objective was to establish a foundational testbed for algorithmic exploration by implementing and adapting a Particle Swarm Optimization (PSO) inspired algorithm, drawing conceptual inspiration from the work of [4] on the interplay between social interaction and agent memory. With preliminary testing of this baseline strategy yielded a modest LOS persistence of only 14.6%. This initial result underscored the inherent difficulty of the problem.

The core focus was to simulate a deployment centric investigation. The core hypothesis was that superior tracking performance could be achieved through a highly adaptive state dependent behavioural configuration within the initially deployed swarm agents. The research aim was to validate an optimal hybrid control system capable of achieving over 90% LOS persistence with the target being twice the speed of an individual swarm agent. The architectural solution of this new approach is a state machine controller implemented in code as `HasMemory_DotProductMode` which delegates tasks to specialized interception, pursuit and search algorithms based on the real time tactical context.

Structure of the Thesis

This thesis documents the full research journey from the establishment of foundational principles to the discovery of a novel, high-performance behavioural configuration. The thesis begins with a comprehensive review of the relevant literature establishing the theoretical context and identifying the specific research gap this work addresses. Following this the Design Requirements and Methodology section details the custom built simulation environment the iterative development and optimization pipeline and the design of the various agent behaviours. The System Design and Implementation sections then provide a deeper look into the modular architecture of the simulator and the evolution of the algorithmic strategies from early prototypes to the final solution. The Results section presents the quantitative outcomes of this process, charting the performance from the initial 14.6% baseline to the final 94.41% LOS persistence through extensive parameter tuning and comparative experiments. Finally, the Discussion chapter interprets these findings and explores their implications for real world applications, while the Conclusion summarizes the

key contributions, acknowledges the limitations of the current work and outlines promising avenues for future research.

8 LITERATURE REVIEW

The challenge of tracking a faster target with a swarm of slower agents lies at the intersection of several key research areas in swarm robotics, including bio-inspired algorithms, force-based motion control and coordinated search strategies. This review surveys the foundational paradigms in the field, identifies the primary sources of inspiration for this project and highlights the specific research gap that this work aims to address. It concludes by examining established search methodologies from related domains that have been adapted to inform the reacquisition behaviours developed in this research.

Foundational Paradigms in Swarm Tracking

Research in swarm robotics has long been dominated by decentralized, heuristic-based algorithms, many of which draw inspiration from collective behaviours observed in nature, such as ant colonies or bird flocks [5]. Comprehensive reviews of the field, such as that by [3], provide a concise overview of its evolution outlining the development of foundational engineered algorithms including Ant Colony Optimization, Artificial Bee Colony and Particle Swarm Optimization (PSO). These surveys adeptly cover the core concepts of emergent behaviour and decentralized control that are central to the swarm paradigm.

However, a recurring theme in this foundational literature is a primary focus on algorithmic theory and simulation based validation. While essential for establishing theoretical viability, these works often lack depth regarding the transition from simulation to hardware implementation [6]. There is frequently limited discussion on the practical challenges such as finite energy constraints, sensor inaccuracies, communication noise, and the effect of environmental obstacles that impede robust real world applications. This gap between algorithmic promise and practical utility underscores the need for research that not only proposes effective strategies but also considers the parameters and contexts that move the field closer to deployment ready solutions.

Among the previously mentioned methods (PSO) a population-based optimization technique introduced by [7], is particularly relevant to this research and forms the conceptual bedrock of our work. In classical PSO individual "particles" navigate a search space, adjusting their trajectory based on their own best known position ("personal best") and the best-known position of the entire swarm (the "global best"). This framework maps naturally to the target tracking problem. The swarm's attraction to a "global best" position can be conceptually translated to the swarm's collective attraction to a target's last known location. This principle fundamentally informs the design of our simulator, where agent motion is governed by a vector sum of attraction and repulsion forces. The continued relevance of PSO as a foundational concept is underscored by recent comprehensive surveys, such as the systematic review by [8] which synthesizes modern PSO variants and hybrid approaches directly applied to target-search and encirclement problems. This conceptual mapping is not merely theoretical, it finds direct application in modern robotics. For instance, the work of [9] demonstrates a practical implementation where PSO-like vector fields are used to guide a formation of Unmanned Aerial Vehicles (UAVs) for target encirclement and persistent observation, providing a direct contrast to the modular, state-switching behavioural architecture developed in our own simulator.

Building on these foundations, a significant body of work has focused on the mechanics of tracking dynamic targets using force based models. In this paradigm, an agent's motion is governed by a vector sum of attractive forces (pulling it towards a goal) and repulsive forces (pushing it away from obstacles or other agents). Early distributed algorithms for multi robot observation of multiple targets laid the groundwork for this approach [1] [10], establishing methods for cooperative control. This force-based logic is a central theme throughout our work, where every agent behaviour is ultimately a composition of attraction, repulsion, and separation forces, aggregated via multipliers and scaled to a maximum speed.

More recently, advanced machine learning techniques, particularly Deep Reinforcement Learning, have been applied to the swarm pursuit problem. State of the art examples, such as the work by [11],demonstrate that Multi-Agent Reinforcement Learning can successfully train decentralized, heterogeneous UAV

swarms for multi-target pursuit, with emergent strategies that can even be transferred to real hardware. Other Studies have demonstrated the use of algorithms like Multi-Agent Deep Deterministic Policy Gradient to enable swarms of slower agents to learn complex, emergent collaborative strategies for pursuing targets [12] [13]. While these methods show great promise in generating highly optimized behaviours, they often rely on centralized training paradigms and make simplifying assumptions about the environment (e.g., full visibility, fixed evader policies). This can limit their direct applicability in noisy, partially observable, real-world scenarios and contrasts with the decentralized, heuristic-based execution model that is the focus of our research. Moreover, deep learning approaches are frequently criticised for their lack of transparency often being treated as "black-box" systems whose internal decision-making processes are difficult to interpret [19]. Our approach prioritizes the development of explainable, parameter-driven behaviours that can be tuned and validated without requiring extensive, model specific training.

Key Behavioural Sub-Problems

While the foundational paradigms provide the algorithmic tools, applying them to the fast-target pursuit problem reveals a set of distinct, interlocking challenges that a successful solution must address. The core technical problem can be decomposed into four primary behavioural sub-problems:

1. **Target Reacquisition:** In a scenario where LOS is frequently lost, the ability to efficiently relocate the target is paramount. Following memory expiration agents must transition from a convergent pursuit strategy to an exploratory search pattern that maximizes the collective sensor coverage of the swarm to minimize the time to reacquisition.
2. **Information Propagation:** Due to the target's superior speed sighting data is a highly perishable asset. In a decentralized architecture, a successful system must facilitate the rapid and robust propagation of new target location information through the swarm's k-nearest neighbour communication network to ensure the collective can react cohesively.
3. **Spatial Distribution:** A common failure mode in swarm tracking is agent clustering, especially around a stale, last known target position. This reduces the swarm's overall sensor footprint and inhibits effective searching. Therefore, agents must maintain an effective spatial distribution, balancing the need for proximity to share information with the need for dispersion to cover more area.
4. **Adaptive Pursuit:** A "one size fits all" pursuit strategy is inherently suboptimal. The tactical approach for an agent positioned ahead of the target (e.g., setting up an intercept) is fundamentally different from that of an agent behind the target (e.g., aggressive catch up to the target). The system must be able to dynamically switch between specialized strategies based on an agent's real time geometric relationship to the target's predicted path.

Addressing these four sub-problems simultaneously within a decentralized framework constitutes the central challenge of this research.

Core Inspirations: Social Interaction and Memory

The primary theoretical inspiration for this project is the research conducted by [4] which directly investigates the challenge of tracking faster targets. Their work utilized a modified PSO inspired approach to demonstrate the crucial interplay between two key parameters: inter-agent communication (defined by the number of k-nearest neighbours an agent interacts with) and agent memory (the duration for which target sighting information is retained, t_mem). Their findings established that tuning these social and memory parameters is essential for balancing the fundamental trade-off between exploration (searching for the target when its location is unknown) and exploitation (converging on the target once it is found).

This framework was directly operationalized in our work through the development of our `SeparationBehavior`. This behaviour serves as our direct adaptation and initial implementation of the core algorithm proposed by [4]. It explicitly models their PSO-inspired logic by computing an agent's velocity as a vector sum of attraction towards the most recent target sighting (exploitation) and repulsion from neighbouring agents to prevent clustering (a form of exploration). The development of this behaviour was our first step in translating the foundational theory into a functional, testable module within our simulator.

Crucially, this baseline behaviour, and indeed every subsequent behaviour developed for the project, exposes `COMMUNICATION_COUNT` and `MEMORY_DURATION_TICKS` as tuneable parameters. This parameter driven methodology, established with our initial adaptation of the [4] model provides the foundation for our optimization framework to systematically explore the exploration exploitation trade-off and discover the optimal balance for each specific tactical role within our final hybrid control system. The `SeparationBehavior` thus represents not only our baseline for performance comparison but also the architectural template upon which more complex and specialized behaviours were subsequently built.

The Unaddressed Gap: Decentralization, Fast Targets and Deployment Strategies

To fully appreciate the research, gap this project addresses, it is crucial to contextualize the problem's inherent difficulty which arises from three distinct layers of complexity.

First, the challenge of multi-agent, multi-target tracking has been compared to NP-Hard problems due to the combinatorial complexity of coordinating assignments and maintaining state with limited information [10]. While our scope is a single target, the core challenges remain. This complexity is fundamentally magnified by our choice of a decentralized control architecture. Unlike a centralized system with access to a perfect, global state and each agent in our swarm operates under a "fog of war." Its decisions are based solely on its own limited sensor data and information shared by a small subset of its nearest neighbours. This constraint introduces significant challenges in information propagation which is often delayed and perishable and requires coordination to emerge from local rules rather than from a top-down command [3].

Second, this architectural difficulty is compounded by the primary task constraint: the target moves at twice the speed of any individual agent. A review of the literature reveals that while target tracking is widely studied, the specific and demanding scenario of a persistently and significantly faster target is less commonly explored. Many studies [1][5][10] assume agents are faster than or at least comparable in speed to, the target. By setting a challenging 2:1 speed ratio, our work intentionally focuses on a "worst-case" scenario where monolithic pursuit strategies are guaranteed to fail, thereby forcing the development of more sophisticated, truly cooperative and adaptive solutions.

Thirdly is the prevalence of homogeneous monolithic behavioural strategies in many foundational heuristic studies [1][4][5]. While an algorithm may be optimized for a specific task like pursuit or area search, it is rare to find frameworks that explicitly and dynamically switch between fundamentally different, specialized strategies in response to changing tactical contexts. A swarm's mission phase can change in an instant from tracking, to losing, to searching, to reacquiring a target. A single set of rules is unlikely to be optimal across all these phases. This suggests a need for adaptive, state-dependent architectures that can delegate control to the most appropriate sub-behaviour at any given moment.

This project positions itself at the intersection of these three challenges. This research explicitly investigates the performance implications of a decentralized system in tracking a significantly faster moving target.

Informing Search and Reacquisition Strategies

A primary challenge in tracking a faster target is reacquisition after LOS is lost. When memory of the target's last known position expires agents must transition from a pursuit or "catch" behaviour to a "search" behaviour. To inform the design of these crucial search algorithms, this project drew inspiration from established search patterns in related but distinct fields, adapting them to serve as specialized modules within our state-dependent framework.

Lévy walks are a class of random walk where step lengths follow a heavy tailed probability distribution, resulting in a mix of short, local steps and occasional long-distance jumps. This strategy is highly efficient for finding sparse targets. The work of [15] extended this to the Biased Lévy Walk, showing that efficiency improves by biasing the walk's direction using a priori information. Our `LevyBiasedStationSearchBehavior` adapts this principle. With no explicit target probability map, we use the agent's "home" base as the source of bias. When memory is lost, the agent performs a Lévy search with its long-distance jumps biased towards its starting position. The selection of a Lévy-based search is supported by recent empirical evidence, such as the study by [16], which demonstrates that Lévy walks remain a competitive strategy for sparse target searches in practical swarm robotics contexts.

However, their work also highlights a critical limitation: the ideal, statistically pure Lévy flight is disrupted by real world factors like collision avoidance. This observation provides strong justification for our implementation of a biased Lévy walk, rather than attempting to replicate a perfect random walk, our model sensibly adapts the strategy by biasing it towards the 'home' base, providing a robust search pattern that avoids aimless wandering while acknowledging the constraints of a multi-agent system.

In the field of multi robot exploration a foundational challenge is for a team of robots to autonomously map an unknown environment. A key concept in this area is the "frontier," defined as the boundary between explored and unexplored space. The work by [17] on Coordinated Multi-Robot Exploration presents a decision theoretic approach where robots are assigned to different points on this frontier. By coordinating to ensure they explore different frontiers the robots avoid redundant work and map the environment much faster than they would by acting independently. The core principle is to identify and move towards gaps in the team's collective knowledge.

This concept was adapted to create our `FrontierGapSeekingBehavior`. In our tracking problem, after losing the target, the "unexplored" space is the area where the target might now be. The positions of an agent's neighbours relative to the target's last known location form a partial perimeter. Our algorithm computes the bearings of all neighbouring agents from this last known position and identifies the largest angular gap between them. This gap represents a "frontier" in the swarm's sensor coverage. The agent then sets its attraction vector towards the centre of this gap. This behaviour directly adapts a concept from robotic mapping to target reacquisition, encouraging agents to spread out and cover areas where the swarm currently has no sensor presence.

Maritime Search and Rescue (SAR) provide a rich, real-world source of established and validated search patterns. The International Aeronautical and Maritime Search and Rescue [2] is the authoritative guide for these operations. It details standardized patterns for scenarios where a target's location is uncertain, including the Expanding Square Search and the Sector Search. These patterns are designed to provide systematic, expanding coverage outward from a datum point (the last known position). This directly inspired our `ExpandingSpiralSearchBehavior`, which provides a simple, computationally efficient spiral search pattern anchored to the last sighting, grounding one of our baseline search algorithms in decades of real-world operational practice.

Further academic work in maritime SAR has sought to optimize these patterns. The research by [18] proposes a novel path finding framework that moves beyond rigid, pre-determined patterns. Their A*-based algorithm dynamically generates paths that optimize for the probability of containment, considering factors like target drift and vehicle kinematics. One of the patterns used as a baseline in their comparison is the Parallel Sweep Search, also known as a "lane sweeper." This provided the direct inspiration and academic justification for our `LaneSweeperBehavior` in which agents upon losing the target, conduct horizontal sweeping motions across the environment to provide systematic area coverage.

This approach of combining and prioritizing multiple search strategies is itself a modern trend in swarm intelligence. For example, hybrid approaches that combine the stochastic properties of Lévy walks with other bio-inspired algorithms, such as the Firefly Algorithm, have been shown to be effective for multi-robot foraging [19]. This supports our methodological decision to build a diverse catalogue of search behaviours, enabling the system to leverage both the systematic coverage patterns from SAR and the strengths of modern hybrid stochastic methods.

Synthesis and Research Position

This project's unique contribution lies in the synthesis of these foundational ideas into a novel, hybrid control system. The architecture is built upon the core PSO-inspired attraction/repulsion grammar, which was first operationalized through `SeparationBehavior`, our direct adaptation of the algorithm from [4]. This behaviour served as our crucial performance baseline and the architectural template for all subsequent development.

Building upon this foundation, this project then introduced a suite of internally developed pursuit focused ("catch") behaviours. These include enhancements such as `PredictiveBehavior` to anticipate target motion, `WedgePredictiveBehavior` to introduce lateral spreading, and

`FollowSightingsBehavior` to create an information gradient within the swarm. These purpose-built algorithms were not drawn from the literature but were designed iteratively to address specific weaknesses observed in the baseline pursuit model.

However, to solve the distinct challenge of target reacquisition when all memory is lost, the project turned to well-established search domains for inspiration. This led to the development of a catalogue of specialized "search" behaviours. These reacquisition strategies adaptations of formal methods from Lévy walk theory `LevyBiasedStationSearchBehavior`, frontier-based exploration `FrontierGapSeekingBehavior`, and maritime search and rescue `ExpandingSpiralSearchBehavior`, `LaneSweeperBehavior`.

Our research, therefore, is positioned to demonstrate that a state dependent architecture, populated with a rich catalogue of both purpose-built pursuit algorithms and principled adaptations of established search strategies provides a robust and high performance solution to the challenging fast target pursuit problem. This hybrid approach, blending direct adaptation, internal designs and spatially separated starting locations, represents a unique contribution to the field.

9 DESIGN REQUIREMENTS & METHODOLOGY

9.1 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

FR1	**2D environment simulation**: The system will simulate a 2D continuous environment where agents and targets will move around in.
FR2	**Agent Kinematic Modelling:** The system will model individual swarm agents as particles capable of movement within the 2D environment, governed by velocity vectors.
FR3	**Target Kinematic Modelling:** The system will model a single target as a particle capable of independent movement within the 2D environment.
FR4	**LOS determination:** The system will be able to determine if an agent has unobstructed LOS to the target within its sensing radius.
FR5	**Inter-Agent Information Exchange:** Agents must be able to share their most recent target sighting data with a limited number of nearest neighbours to improve collective awareness.
FR6	**Simulation Visualisation:** The system shall provide a visual representation of the environment, agents and target allowing for real time observation of their states and interactions

Table 1 Functional Requirements

NFR1	**Observability & Debuggability:** The simulation environment must be sufficiently observable to allow for qualitative assessment of agent behaviours and aid in debugging algorithmic implementation.
NFR2	**Modularity:** The system architecture must separate key components (Behaviours, Paths, Starting Locations, Optimization, Agents, etc.) to allow for easy testing and swapping of new strategies.
NFR3	**Configurability:** The simulation will allow for basic configuration of key parameters (e.g. agent speed, target speed, sensor range) to facilitate testing and observation.
NFR4	**Version Control:** All source code and relevant project files will be managed using a distributed version control system (Git/Github) to ensure collaborative development, change tracking and code integrity.
NFR5	**Platform Support:** The core simulation will be executable on all platforms supportive of Python and Pygame.

Table 2 Non-Functional Requirements

9.2 DESIGN CRITERIA AND CONSTRAINTS

DC1	**Target LOS Maintenance**: The swarm will maintain LOS with the target for at least 90% of the total simulation time during tracking

DC2	**Agent Resource Minimisation:** The proposed multi-base deployment strategy will achieve the target LOS maintenance (DC1) using a minimized average and peak number of active agents.
DC3	**Fast Target Tracking:** The system must effectively track a target moving exactly two times the speed of any single agent.
DC4	**State base behaviour:** The final solution must dynamically assign different behaviours based of acquired information.
DC5	**Separate Take-off locations:** The agents must use separated take off locations

Table 3 Design Criteria

C1	**Non-Evasive Target:** The target must be non-evasive (its movement is fixed by its path, it does not react to agents).
C2	**Simulation Environment**: The project is to be developed and validated entirely within the 2D simulation environment. No physical hardware implementation is planned.
C3	**Development Tools:** The primary development language is Python, leveraging the Pygame library for simulation and visualisation due to its rapid prototyping capabilities and accessibility.
C4	**Resource Limitations:** Development relies on personal computing resources available to the team members. There is no dedicated budget for software or hardware.
C5	**Simplified World Model:** The initial simulation assumes perfect communication within k-nearest neighbours (when attempted), perfect sensing within range and an obstacle free environment to simplify initial algorithm development and analysis.
C6	**Simulation Boundaries:** All members of the 2D environment are unable to leave the bounds of 1280x720 unit simulation space.

Table 4 Constraints

Constraint #	Constraint/Criteria	Value
C7	**Agent Sensing Range:** Agents must have a limited Line of Sight (LOS) range.	R_{LOS}=25 units
C8	**Agent Speed:** Maximum speed constraint for individual agents	V_{Agent}=1unit/tick
C9	**Target Speed:** Fixed speed for the target	V_{Target}=2 unit /tick
C10	**Test Duration:** How long each simulation lasts.	T_{sim}=1000 ticks

Table 5 Value Based Constraints

9.3 METHODOLOGY USED

The project followed a rigorous, 10-step **Iterative Design and Optimization Methodology** designed to systematically develop, tune, and validate the swarm's behavioural strategies against the demanding fast-target tracking requirement.

The methodology relied heavily on the **Tester** and **Optimizer** classes (implemented in testing.py) and the **Sequential Search** strategy (optimization_strategies.py) to automate performance scoring (LOS Persistence %) and parameter optimization across all predefined target paths (paths.py).

Iterative Development and Refinement Cycle

1. **Create Foundational Simulation Environment:** Developed the core 2D vector-based simulation environment using **Python 3.12.9** and **Pygame** (simulation.py)
2. **Implement Basic Foundational Behaviours:** Defined simple behaviours (e.g., SeparationBehavior, PredictiveBehavior) that serve as building blocks and initial concepts (agent_behaviors.py).
3. **Establish Basic Optimization Framework:** Integrated the **Sequential Search** algorithm (optimization_strategies.py) to systematically tune the parameters of any given behaviour class by maximizing the LOS Persistence Score across all target paths.

4. **Optimize Basic Behaviours:** Used the optimization framework to find the best possible parameter settings (e.g., memory duration, prediction multipliers, etc) for each foundational behaviour.
5. **Test to Get Benchmark LOS % Scores:** Ran the optimized foundational behaviours against a battery of diverse target paths to establish a low-end performance **benchmark**.
6. **Repeat Steps 2, 3, 4, 5 for All New Behaviours:** Iteratively designed and tested new, more sophisticated behaviours (e.g., PredictiveInterceptMultiplierBehavior, etc) to build a library of high-performing, general purpose and specialized behaviours.
7. **Test State Machine Switching Logic:** Took the best-performing behaviours from the library and integrated them into the **StateMachineBehavior** class (agent_behaviors.py). Initial tests focused on developing the best transition logic (i.e., the rules for switching between states.
8. **Alternate Steps 6 and 7 to Refine State Switching Logic:** Repeated the design/testing process, where the results from state-machine testing informed the necessary refinements to the individual sub-behaviours and, conversely, refined behaviours improved the overall state machine performance.
9. **Find Best State Switching Logic and Behaviours and Optimize Them Together:** Once a promising hybrid configuration was found, the final, high-level parameters of the StateMachineBehavior (e.g., state transition thresholds, collective memory duration) were jointly optimized with the refined sub-behaviour parameters.
10. **Repeat Step 9 until Best Overall Setup is Found:** This final, hyper-iterative optimization phase was used to find the global optimum, resulting in the final high-performance strategy that achieved the target LOS Persistence Score of 94.41%.

9.4 TOOLS/SOFTWARE USED

Category	Tool/Software	Purpose
Core Language	**Python 3.12.9**	Primary programming language for the simulation and testing framework.
Simulation Engine	**Pygame**	Used for vector mathematics, 2D simulation rendering, and time management.
Integrated Dev. Env	**VS Code**	The primary development environment for writing, debugging, and managing the Python source code.
Documentation	**MS Word**	Used for writing and editing, research paper & final project report.

10 SYSTEM DESIGN AND DEVELOPMENT

10.1 CONCEPTUAL DESIGN

The project's conceptual design is rooted in the Strategy Design Pattern to ensure the system is flexible and highly modular for experimentation. Figure 1, the top-level logic flow, demonstrating the deliberate

separation of the core simulation loop (the "what") from the complex decision-making algorithms (the "how"). This design choice facilitates rapid testing of different agent behaviours and target path scenarios without modifying the fundamental physics engine. The flow begins with the global configuration being passed to the main simulation entities, which then delegate their movement and decision-making responsibilities to swappable logic modules, allowing for state-dependent and dynamic behaviour changes within the system.

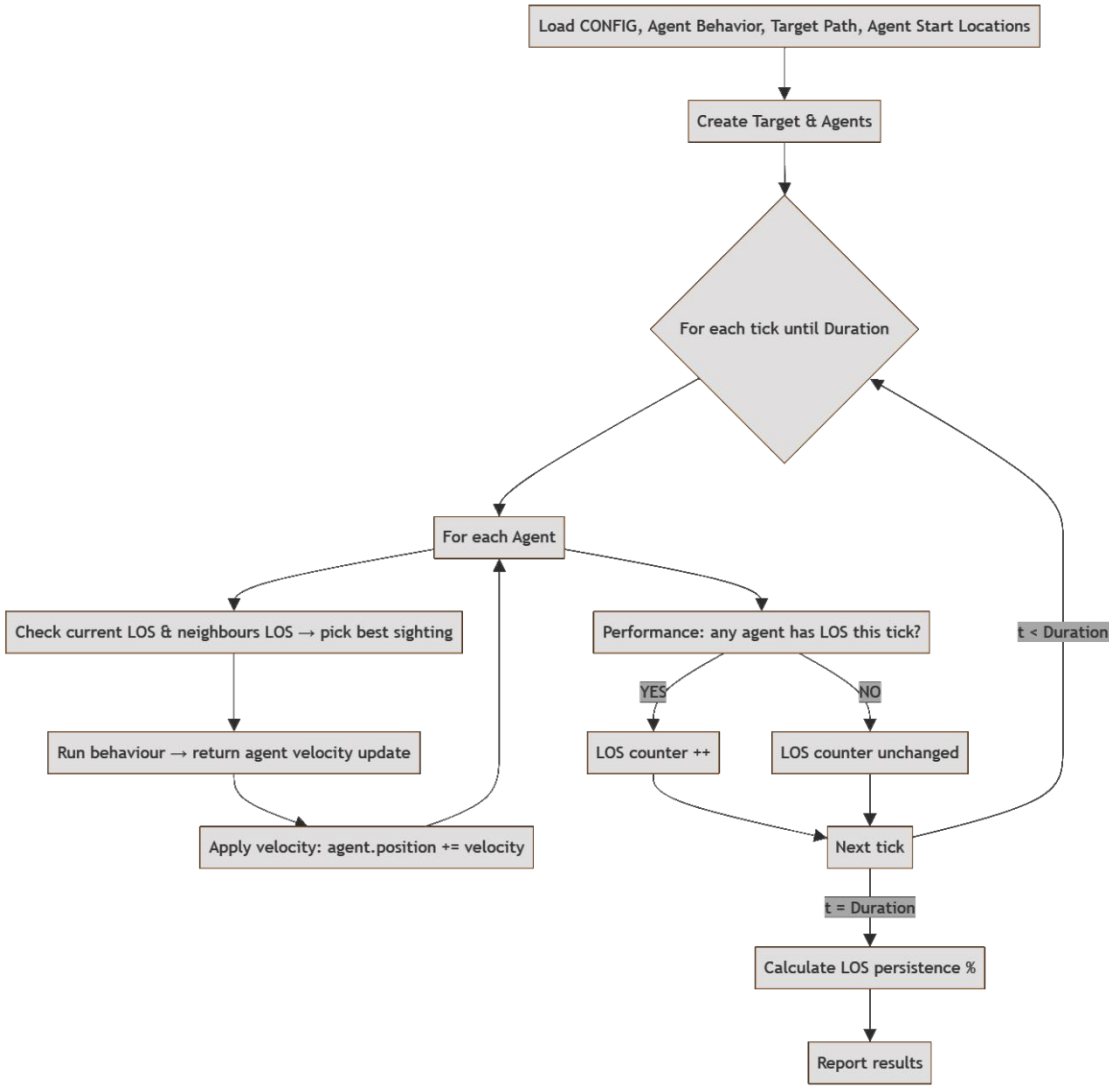

Figure 1 High Level System Architecture Flow Chart

10.2 System Architecture and Detailed Design

The simulation framework is built on a modular, object-oriented architecture in Python, designed to separate high-level testing and optimization from the core simulation engine and agent-specific logic. This separation allows for rapid prototyping and validation of different swarm strategies. The system's design can be understood by examining its four primary layers: the Testing and Optimization Framework, the Core Simulation Engine, the Modular Logic Components (Strategies), and the Utility Module.

10.2.1 Testing and Optimization Framework

This layer is the outermost shell of the application, responsible for configuring and running experiments.

- **`main.py` (Entry Point):** This script serves as the main entry point. It defines the global `CONFIG` dictionary (containing parameters like screen size, agent count, and speeds) and selects the high-level components for a specific experiment (e.g., `SHARED_AGENT_BEHAVIOR`, `SHARED_TARGET_PATHS`). It initiates one of the three run modes: visual simulation, a single headless test, or a full optimization.
- **Optimizer (`testing.py`):** This class acts as a high-level wrapper for running a full optimization experiment. It is initialized with the base `CONFIG` and the classes to be tested. Its primary role is to instantiate and delegate the complex optimization loop to a specific OptimizationStrategy class.
- **OptimizationStrategy (`optimization_strategies.py`):** This is an abstract base class for different tuning algorithms.
    - **Concrete Strategies (e.g., `RandomSearch`, `SequentialSearch`):** These classes contain the main optimization loop. They systematically generate new sets of parameters (e.g., `MEMORY_DURATION_TICKS`, `ATTRACTION_MULTIPLIER`) and, for each set, they call the Tester class to run a full simulation and retrieve a performance score. The SequentialSearch strategy, for example, iterates through each tuneable parameter one by one, running parallel trials to find the optimal value for that single parameter before moving to the next.
- **Tester (`testing.py`):** This is the simplest execution wrapper. Its sole responsibility is to be initialized with a complete configuration (agent behavior, target path, and all parameters), instantiate a single Simulation object, run the simulation in headless mode (run_headless), and return the final performance score.

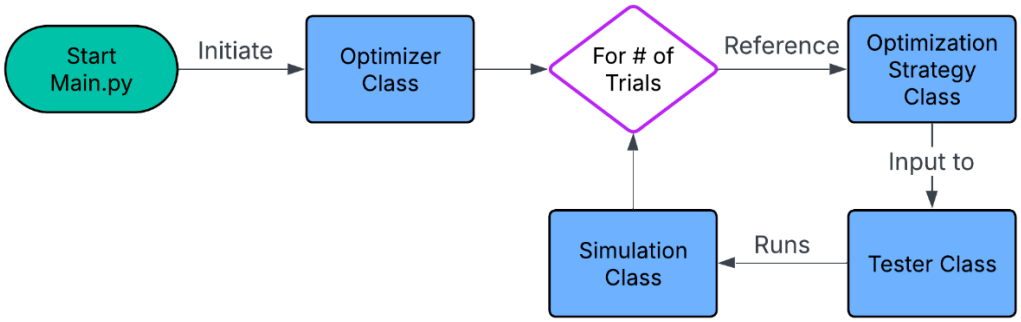

Figure 2 Testing Framework Flowchart

10.2.2 Core Simulation Engine (`simulation.py`)

This layer is the heart of the application, managing the state and time of a single simulation run.

- **Simulation Class:** This class orchestrates the entire simulation world.
    - **Initialization:** It is initialized with the `CONFIG`, an agent_behavior_class, and a target_path_class. During its _setup(), it instantiates all the necessary components:
        1. It instantiates a `PathStrategy` (from `paths.py`) and passes it to a new Target object.
        2. It reads the `STARTING_STRATEGY` from the agent_behavior_class (e.g., `TriangleTilingLocation`), instantiates it, and calls `generate_positions()` to get a list of starting coordinates.

3. It creates and stores a list of Agent objects, providing each one with its starting position and the agent_behavior_class
    - **Main Loop**: Its step() method is called once per simulation tick. It advances the simulation by calling `self.target.update()` and `agent.update()` for every agent. It also tracks the performance metric (Line-of-Sight) by checking `agent.last_sighting_time`.
- **Agent Class:** This class represents a single agent in the swarm.
    - **Initialization:** Upon creation, it instantiates its own behaviour object: self.behavior = behavior_class(config). This is a key design choice (the Strategy Pattern): the Agent class does not know how it will move, it only knows that it has a behaviour object that will tell it how to move.
    - **Update Logic:** Its update() method is called by the Simulation class. This method delegates the complex decision-making by calling self.behavior.calculate_velocity(self, all_agents, target, current_tick). It then takes the returned velocity vector and applies it to its own position.
- **Target Class:** This class represents the target.
    - **Initialization:** Like the Agent, it uses the Strategy Pattern. It is initialized with a path_strategy object.
    - **Update Logic:** Its update() method is called by the Simulation. It simply delegates the movement logic by calling self.path_strategy.update() to get its new position for the current tick.

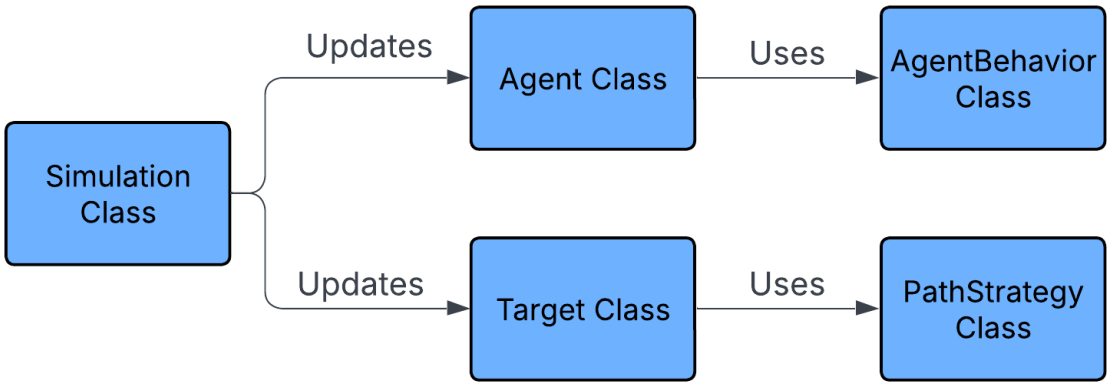

Figure 3 Simulation Class Interaction Flowchart

10.2.3 Modular Logic Components (The "Strategies")

This layer contains the swappable "brains" for the agents and target, allowing for easy experimentation.

- **`agent_behaviors.py`:** This is the most complex logic module.
    - **AgentBehavior (Base Class):** An abstract class that defines the interface, primarily the calculate_velocity() method.
    - **Concrete Behaviors (e.g., `SeparationBehavior, PredictiveBehavior`):** These classes implement the actual calculate_velocity() logic. They are fed the full state (agent, all_agents, target, current_tick) and perform vector calculations to return a single velocity vector.
    - **State Machine (HasMemory_DotProductMode):** This is a special hybrid behaviour. It also implements the AgentBehavior interface, so the Agent class can hold it just like any other behavior. However, its own `calculate_velocity()` method does not perform vector math. Instead, it acts as a controller, checking the agent's state (e.g., "Do I have

memory?", "Is the dot product positive?") and then delegating the call to another, specialized behaviour (like `GetInFrontBehavior` or `InverseSquareRepulsionBehavior`). This creates a state-dependent, multi-layered strategy.

- **`paths.py`:**
  - **PathStrategy (Base Class):** An abstract class defining the update() interface.
  - **Concrete Paths (e.g., `WaypointPath, HorizontalPatrolPath`):** These classes implement the update() method, returning a new Vector2 position for the target each tick.
- **`starting_locations.py`:**
  - **StartingLocationStrategy (Base Class):** An abstract class defining the generate_positions() interface.
  - **Concrete Strategies (e.g., `TriangleTilingLocation, GridLocation`):** These classes implement the generate_positions() method, returning a list of Vector2 coordinates used by the Simulation class during setup.

10.2.4 Utility Module

- **`helper_functions.py`:** This is a stateless collection of pure functions. It includes common, reusable calculations like `find_nearest_neighbors()` and `get_best_sighting()`. This module is primarily imported by agent_behaviors.py to avoid code duplication and keep the behaviour logic clean and readable.

11 IMPLEMENTATION

11.1 PROTOTYPING

Early development focused on establishing a minimal, yet functional agent behaviour inspired by basic particle swarm optimization (PSO) strategies and existing research [7][4]. The first implementation was the **`SeparationBehavior`**, a purely formulaic approach which had no state switching and simply combined two weighted vectors to determine its new position each time step. This included an attractive vector towards the best-known target location available withing the **"`MEMORY_DURATION_TICKS`"**, which was stated in the behaviour parameters. And a repulsion vector which moved uniformly away from the k nearest neighbouring agents, where k was also defined in the behaviour parameters as **"`COMMUNICATION_COUNT`"**. Lastly the **"`ATTRACTION_MULTIPLIER`"** defined the relative strength of attraction compared to repulsion allowing for further us to tune the behaviour for an optimal amount of agent clustering. Despite its simplicity this behaviour was very effective and was used as a benchmark throughout our whole project. Our initial testing scored a modest 14.6% LOS baseline but was ultimately limited by the simulation architecture and spawn locations.

11.2 TESTING FRAMEWORK

The single most important implementation effort was the creation of an automated testing and optimization framework (`testing.py, optimization_strategies.py`). The development of this framework was directly motivated by a recognized gap in the swarm robotics literature. Research by [6] for instance, highlights the lack of standardized test environments needed to fluently test and compare multiple swarm algorithms and their parameters. Our framework was designed to be such an environment, providing the modular and automated testbed that proved to be. This framework was the driving force behind our radical improvement from 14.6% to 94.41% LOS persistence.

The testing and optimization framework can be broken down into two parts.
Tester Class: a simple class that takes a config of the simulation state being tested (i.e., agent behaviour, agent behaviour parameters, list of target paths) and runs headless tests on each target path with the provided simulation parameters. It returns the average LOS% score between all the paths in the **"`SHARED_TARGET_PATHS`"** list as the result for the given test.

OptimizationStrategy (Base Class): An abstract class that defines the interface for all optimization strategies. Its job is to automatically find tuneable parameters withing the selected agent behaviour and modify them according to the headless test results and the selected optimization strategy.

Together the **Tester and OptimizationStratagy** run multiple tests and tune the behaviour params to hopefully arrive at an optimal set of behaviour params that achieve a local or global best LOS% score.

In particular there were 2 optimization strategies that did 99% of the heavy lifting throughout all our testing. This was the **RandomSearch** strategy and its successor the **SequentialSearch** strategy. In Particular these two strategies won out over typical gradient descent approaches because of their ability to run multiple parallel tests which allowed us to run 20+ Tester simulations at a time, drastically improving the speed at which behaviours could be optimized and increasing the complexity of behaviour that we could reasonably explorer.

RandomSearch: Worked by taking the initial behaviour parameters and for every test simply nudging each value up or down by a random amount determined by the "search_factor" variable. It would keep track of the best set of behaviour parameters it found through all of its tests and return these to the user once testing has completed. Limitations with this strategy lie in the fact that it was permanently bound within 1 search factor of the initial parameter values, and to escape this you had to stop the optimization, manually adjust input the best params and rerun a new optimization centred around the new parameter values. Additionally, the randomsearch would change all the parameters every trial, making it unlikely to refine existing successful parameters variations.

SequentialSearch: Aimed to rectify the shortcomings of its predecessor allowing for complete set and forget optimization the sequentialsearch strategy was used in the later half of the 2nd semester to optimize our strongest solutions and ultimately helped us arrive at out final behaviour. Its operation can be broken down into three main improvements over the random search. Firstly, it performed random searches on one parameter at a time for a given number of trials, this allowed good parameter combinations to be fully explored and made into great combinations. Secondly, it updated the initial parameters after each individual parameter optimization ensuring each sequential search was centred around the most successful set of parameters every time. This allows the optimization to explore 100% of the solution space with zero human intervention. Lastly, the sequential search had the ability to run through the list of parameters multiple times, this is necessary because changing one parameter often alters the optimal value for all the other parameters.

This framework was integral to the success of our project and allowed for levels of optimization and testing that would have been previously impossible due to time constraints.

11.3 TARGET PATHS TESTED

Considering we loosely framed our simulation around a situation where a swarm of police pursuit drones are trying to track a fast vehicle, our target paths were designed to mimic possible route a car could take. Considering our simulation screen is scaled to be approximately 10 to 15km wide, and the agents (drones) have a sight radius of about 200-300m in this scale we decided minor evasive manoeuvres could be ignored. With this assumption we could model the vehicles overall heading with straight target paths that traversed the screen over the length of the simulation. These paths were designed to be different from one another, while stressing different aspects of the behaviours to avoid over specialized characteristics emerging.

Waypoint 1 Path: (Top-Left) → (Top-Right) → (Bottom-Left) → (Bottom-Right) → Repeat

Waypoint 2 Path: (Top-Left) → (Bottom-Left) → (Top-Right) → (Bottom-Right) → Repeat

Waypoint 3 Path: (Middle) → (Top-Middle) → (Top-Right) → (Bottom-Right) → (Bottom-Left) → (Top-Left) → Repeat

Horizontal Patrol: (Middle-Left) → (Middle-Right) → Repeat

11.4 EXPLORED SOLUTION

`SeparationBehavior` (V2): Building off the original SeparationBehavior this version added a MINIMUM_DISTANCE variable. If two agents were close than this minimum distance the separation_vector would become very large and act to separate the two agents with a very strong vector force.

`PredictiveBehavior`: Building off the SeparationBehavior (V2), the predictive behaviour is not attracted to the targets location but instead attracted to where the target is going to be in the future. It achieves this by first finding the time it would take to travel to the targets current position. It then uses this time as a multiplier for how far ahead of the targets current position it should aim its attraction vector.

`WedgePredictiveBehavior`: This adds one feature over the predictive behaviour method. When moving towards the targets future position, the agent is constantly checking whether there is another agent within its CLUMPING_THRESHOLD radius. If there is, a force perpendicular to the agent's attractive vector and away from the near neighbour is created and scaled according to the LATERAL_PUSH_STRENGTH parameter.

`FollowSightingsBehavior`: Inspired by the separation behaviour, this only differs when the agent has information about the target received from a neighbouring agent. In this case it is attracted to a point between he known target position and the position of the neighbour who attained the target information.

`ExpandingSpiralSearchBehavior`: When this behaviour has information about the target position it will begin to spiral around this location until the information is too old, or it spots the target. On the other hand, if it does not have any information about the target position it will begin to spiral around its starting location in an attempt to spot the target.

`LaneSweeperBehavior`: When this behaviour has information about the target position it will begin to sweep horizontally about this location until the information is too old, or it spots the target. On the other hand, if it does not have any information about the target position it will begin to sweep horizontally about its starting location in an attempt to spot the target.

`LevyBiasedStationSearchBehavior`: A levy step can be classified as a random small or large step in a random direction; the agent will pick a location one levy step away from itself and continue to move towards it each time step. Once it reaches the location it will pick a new location using the same method. The LevyBiasedStationSearchBehavior work differently depending on the kind of information it has. If it has information about the target's positions it will bias the levy step away from the target using the OUTWARD_BIAS_RATE parameter variable. Additionally, if it does not have information about the target it will move to its home location and then being to move according to sequential unbiased levy steps.

`FrontierGapSeekingBehavior`: If you draw a line between self and each neighbouring agent and then consider each of the segments these lines create radially around self. The gap_dir vector is defined by the direction towards the radial gap that is closed to the direction between self and target while still being larger than `MIN_MEANINGFUL_GAP` parameter variable. The attraction vector is a weighted sum of a vector between self and target and the gap_dir variable.

`RadtioMode` (State-Machine): This was the first state switching solution we experimented with and worked as follows. Each agent would talk to their nearest neighbours and check what state each of them were using and then would swap to whatever state it finds the least of in its immediate neighbourhood. Alone this leads to agents changing state every single time step as most ratio values are nearly impossible to achieve perfectly, thus we introduced a `MIX_RATIO_TOLERANCE` which defined how much a neighbourhood could stray from the ideal ratio before it forced state changes.

11.5 Final Solution

The final high-performance solution achieved a 94.41% average LOS score over all target paths. It was implemented in the agent_behaviors.py script and can be broken down into 3 behaviours and 1 state machine.

`HasMemory_DotProductMode` (State Machine): The state machine selected between the three solution behaviours depending on two key deciders. Firstly, it checked whether the agent had information about the target, this can be obtained either by personally having line of sight with the target, within the last `MEMORY_DURATION_TICKS` number of time steps. Or one of the agents k nearest neighbours having line of sight with the target within its `MEMORY_DURATION_TICKS` number of timesteps. Most recent sighting from all obtained information is selected for the agent in question. If the agent cannot find any information that is recent enough to work off it will choose to use the `InverseSquareRepulsionBehavior` (ISR) behaviour.
Alternatively, if the agent does have recent sighting information to work off it needs to check whether it is in front of the target with respect to its movement direction or behind. This is obtained by checking the dot product between the targets move direction and the agent's relative position with regards to the target. If this is positive the agent should use the `GetInFrontBehavior` (GIF) behaviour and if it is negative the `PredictiveInterceptMultiplierBehavior` (PRED).

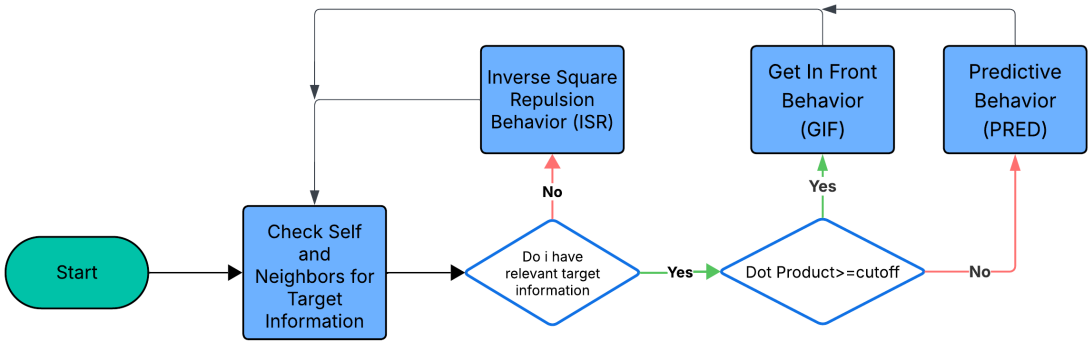

Figure 4 Final Solution State Transition Flowchart

`GetInFrontBehavior` (GIF): The GIF behaviour can be broken down into two parts, the lateral alignment phase and the pursuit phase. Additionally, the GIF behaviour is unique because its inter-neighbour repulsion can only parallel to the target's movement direction. This is important because the lateral alignment phase requires unrestricted movement perpendicular to the target move direction to be effective.

> **Lateral Alignment (LAT):** Moves perpendicular to the target movement direction, attempting to position itself directly in front of the target.
>
> **Pursuit (PUR):** Once directly in front of target, travels straight towards the target.

GetInFrontBehavior (GIF):

```
Key Parameters:

        GIF_MEMORY_DURATION_TICKS: How long to remember a sighting.
        GIF_COMMUNICATION_COUNT: How many neighbours to check for info.
        GIF_MINIMUM_DISTANCE: Separation distance from other agents.
        GIF_SEPARATION_FORCE: Multiplier for the separation force.
        GIF_LATERAL_OFFSET: How far (in pixels) the agent can be from the
        target's path line before it stops trying to move sideways and starts
        pursuing.

FUNCTIOIN:

        LATERAL_DEVIATION = Perpendicular distance between agent and target
direction
        IF LATERAL_DEVIATION IS GREATER THAN LATERAL_OFFSET
                // Phase 1: Not aligned. Move sideways to get onto the path
line.
                ATTRACTION_VECTOR = Perpendicular to target path (Vector)
        ELSE:
                / / Phase 2: Aligned. Move directly towards the target's
position.
                ATTRACTION_VECTOR = Directly towards target (Vector)
        END IF

        FOR EACH NEIGHBOUR:

                IF NEIGHBOR CLOSER THAN MINIMUM_DISTANCE

                        SEPARATION_VECTOR+= Away from Neighbour (Vector)

                END FOR

        END IF

        SEPARATION_VECTOR= SEPARATION_VECTOR (Parallel to target movement
Direction)

        FINAL_DIRECTION=ATTRACTION_VECTOR + SEPARATION_VECTOR * SEPARATION_FORCE
```

`PredictiveInterceptMultiplierBehavior` (PRED): The PredictiveInterceptMultiplierBehaviour is functionally the same as the PredictiveBehaviour with the

addition of a multiplier constant that is added to the attraction vector when there is memory of the target.

```
PredictiveInterceptMultiplierBehavior (PRED):

Key Parameters:

        pred2_MEMORY_DURATION_TICKS: How long to remember a sighting.
        pred2_COMMUNICATION_COUNT: How many neighbours to check for info.
        pred2_MINIMUM_DISTANCE: Separation distance from other agents.
        pred2_ATTRACTION_MULTIPLIER: The base attraction strength.
        pred2_AHEAD_BONUS_MAX: The flat bonus added to attraction when "ahead".

FUNCTIOIN:

        // 1. Prediction and Base Attraction
        TIME_TO_INTERCEPT = distance_to_target / MAX_SPEED
        PREDICTED_POS = TARGET_DATA.position + (TARGET_DATA.velocity *
        TIME_TO_INTERCEPT)
        ATTRACTION_VECTOR = PREDICTED_POS - agent.position

        // 2. Determine Attraction Weight (Base + Bonus)
        ATTRACTION_WEIGHT = pred2_ATTRACTION_MULTIPLIER
        IS_DIRECTLY_VISIBLE = check_line_of_sight(agent, target)
        IS_AGENT_AHEAD = calculate_projection_onto_path(agent, TARGET_DATA) > 0



        IF NOT IS_DIRECTLY_VISIBLE AND IS_AGENT_AHEAD:

                // Apply speed bonus to cut off the target
                ATTRACTION_WEIGHT = ATTRACTION_WEIGHT + AHEAD_BONUS_MAX

        END IF



        // 3. Calculate Separation Force
        FOR EACH NEIGHBOUR:

                IF NEIGHBOR CLOSER THAN MINIMUM_DISTANCE:

                        SEPARATION_VECTOR += Away from Neighbour (Vector)

                END IF

        END FOR

        // 4. Combine and Move
        FINAL_DIRECTION = (ATTRACTION_VECTOR * ATTRACTION_WEIGHT) +
        SEPARATION_VECTOR
```

`InverseSquareRepulsionBehavior` (ISR): The ISR behaviour works by looping over all neighbouring agents and creating a repulsion vector proportional to the inverse square distance between self and neighbour. Additionally it has a small force attracting agents back to their starting locations to

prevent as our testing proved this was effective.

```
PredictiveInterceptMultiplierBehavior (PRED):

Key Parameters:

        ISR_COMMUNICATION_COUNT: How many neighbours to check for info.
        ISR_HOME_WEIGHT: Multiplier for the neighbour repulsion force.
        ISR_SPEED: A percentage scalar for the final speed.

FUNCTIOIN:

        // 1. Calculate Repulsion Force (Inverse Square Law: 1/r^2)

        FOR EACH NEIGHBOUR:

                DISTANCE = distance(agent.position, neighbor.position)
                FORCE_MAGNITUDE = 1.0 / (DISTANCE ^ 2)
                // Sum forces, pushing away from neighbor
                REPULSION_FORCE += direction_AWAY_from_neighbor *
                FORCE_MAGNITUDE

        END FOR

        // 2. Calculate Home Attraction Force (Squared Distance Law: r^2)
        DISTANCE_FROM_HOME = distance(agent.position, agent.starting_position)
        // Force is proportional to distance^2
        FORCE_MAGNITUDE = DISTANCE_FROM_HOME ^ 2
        // Apply force, pulling toward home
        HOME_ATTRACTION_FORCE = direction_TOWARD_home * FORCE_MAGNITUDE

        // 3. Combine and Move
        FINAL_DIRECTION = (REPULSION_FORCE * HOME_WEIGHT) +
        HOME_ATTRACTION_FORCE
```

11.6 CHALLENGES ENCOUNTERED AND SOLUTIONS

Slow and labour-intensive parameter optimization

1. The first issue we encountered was regarding the tuning of behaviour parameters. Initially there was no system in place to tune them, and the process was being done very slowly by hand, changing one value at a time. As the behaviours became more complex this approach quickly became too time intensive. Thus, we developed the optimization framework which would automatically refine the parameters.
2. While the optimization framework was orders of magnitude quicker at tuning behaviours it still was taking too long for extremely complex behaviours. This is when we implemented parallel optimization, utilising every core of the CPU to run multiple optimization sessions at once.
3. Even though our optimization was working very fast, it still required a person to constantly update the parameter values with the ones the optimizer calculated. The last solution we needed was an automatic way to update the parameter values with the best-found ones in a continues fashion. Once this was implemented you could truly walk away from the computer and let the optimization run its course and return with certainty, they parameters would be optimal when you returned.

Hyper Specialized Solutions

- Initial testing and optimization were conducted on a single target path. When we inevitably decided to test the behaviours on alternate paths, we quickly realized that while they performed astonishingly well on the path, they were trained on they performed incredibly poor on every other path. This is what lead to us average the LOS% score amongst a number of paths, ensuring they resulting solutions were robust and capable in all scenarios.

12 RESULTS

This section represents the empirical results of the research, detailing the performance of various behaviour configurations as evaluated within the custom-built swarm simulator. The chapter is structured to mirror the iterative development and optimization cycle, beginning with the establishment of a performance baseline and progressively building towards the final, high performance hybrid solution. The primary metric for evaluation throughout is Line-of-Sight (LOS) Persistence %, which quantifies the percentage of simulation of simulation ticks where at least one agent has a direct view of the target, averaged over multiple Monte-Carlo trials.

Experiment Methodology

All experiments were conducted using a modular, 2D vector based simulation environment developed for this project. The framework implemented in Python 3.12.9 with the Pygame library was designed for rapid repeatable and statistically robust testing. For all reported results the simulation parameters were held constant to ensure a fair comparison:

- **Swarm Size (N_{agents}):** 30 agents. This number was established through preliminary scalability tests to identify the minimum swarm size required to consistently achieve the >90% LOS performance goal. Given the 10 starting locations, this configuration corresponds to 3 agents per station, which was found to be the minimum effective deployment for the 2:1 speed deficit scenario.
- **Target Speed (V_{target}):** 2.0 units/tick
- **Agent Speed (V_{agent}):** 1.0 units/tick (a 2:1 speed deficit)
- **Sensing Range (R_{sight}):** 25 units
- **Target Paths:** A standardized suite of four non evasive paths (WaypointPath1, WaypointPath2, WaypointPath3, HorizontalPatrolPath)

Key accomplishment of the developed work was an automated optimization framework (`testing.py`, `optimization_strategies.py`). This framework was integral to tuning the numerous parameters (e.g., `MEMORY_DURATION_TICKS, ATTRACTION_MULTIPLIER`) for each behaviour. The methodology relied heavily on the `SequentialSearch` strategy, which proved superior to both `RandomSearch` and traditional gradient descent methods. Its ability to perform parallelized, variable by variable refinement allowed for deep exploration of the parameter space, ensuring each behaviour was tested at its optimal or near optimal configuration.

The discovery of the final behavioural configuration followed a rigorous, multi-phase evaluation pipeline:

- **Phase 1: Single-Behaviour Screening**: Each individual "catch" behaviour was optimized and benchmarked to establish a performance hierarchy.
- **Phase 2: Hybrid Pairing:** The top performing behaviours were integrated into the HasMemory_DotProductMode state machine to test the efficacy of a two state (ahead/behind) hybrid strategy.
- **Phase 3: Tertiary Sweep:** The best-performing pair was augmented with a dedicated "search" behaviour for the no memory state to create a three state hybrid strategy.
- **Phase 4: Exploratory Design:** In parallel, new behaviours were designed and implemented specifically to address tactical weaknesses observed during the systematic testing phases.

Phase 1: Baseline Performance of Single Behaviours

The initial benchmark for this project was the 14.6% LOS persistence achieved from the baseline algorithm implementation of the heavily inspired adaption of the PSO strategy from [5]. The first step in Semester 2 was to optimize this behaviour and test it alongside a new catalogue of internally developed "catch" behaviours. After extensive tuning (approx. 500 trials per behaviour), the following performance hierarchy was established.

Behaviour Name	Description	Average LOS%
PredictiveMultiplierBehavior	Predictive pursuit with an "ahead" bonus multiplier.	**55.63%**
PredictiveBehavior	Estimates target's future position for intercept.	**50.45%**
FollowSightingsBehavior	Moves toward the neighbour with the freshest sighting.	**45.79%**
SeparationBehavior	Baseline attraction/repulsion model.	37.61%
WedgePredictiveBehavior	Predictive pursuit with lateral spreading.	34.50%
FrontierGapSeekingBehavior	Moves into the largest gap in swarm coverage.	26.07%

Table 66 Performance of Optimized Single Behaviours

The results in **Table 1** yield a critical insight: predictive algorithms (`PredictiveMultiplier` and `PredictiveBehavior`) dramatically outperform non-predictive ones. The `PredictiveMultiplierBehavior`, which is functionally the same as the `PredictiveBehaviour` but it multiplies the attraction vector by a multiple when tracking a target emerged as the strongest monolithic strategy. However, even with a nearly four-fold improvement over the initial baseline a ~56% LOS score was well below the research goal, confirming the hypothesis that no single behaviour would be sufficient. Based on these results, `PredictiveMultiplier`, `PredictiveBehavior` and `FollowSightingsBehavior` were promoted to the next phase of testing.

Phase 2: Performance of Hybrid Paired Behaviours

This phase investigated the performance of a two-state hybrid system using the `HasMemory_DotProductMode` state machine. This controller assigns different behaviours based on the agent's geometric position relative to the target's velocity vector (ahead or behind). The top-performing single behaviours were exhaustively paired against each other. Also note that the "less-than" (behind) behaviour also served as the fallback for the no-memory state.

Greater-Than (Ahead) Behaviour	Lesser-Than (Behind) / No-Memory Behaviour	Average LOS%
FollowSightingsBehavior	**PredictiveBehavior**	**68.19%**
FollowSightingsBehavior	PredictiveMultiplierBehavior	66.76%
SeparationBehavior	PredictiveMultiplierBehavior	64.57%
PredictiveBehavior	PredictiveBehavior	55.79%

PredictiveBehavior	FollowSightingsBehavior	53.50%
FollowSightingsBehavior	SeparationBehavior	42.30%
SeparationBehavior	PredictiveBehavior	32.64%

Table 77 Performance of Optimized Paired Behaviours

The data in **Table 2** clearly demonstrates the power of adaptive, state dependent control. The top two configurations Predictive + FollowSightings and PredictiveMultiplier + FollowSightings, both achieved a significant performance uplift of over 10 percentage points compared to the best single behaviour. This validates the core concept of the state machine: assigning specialized roles is more effective than a one-size-fits-all approach. To break the tie, a large-scale validation run of ~4,800 trials confirmed that the Predictive + FollowSightings pairing was marginally stronger and more stable.

Phase 3: Performance of Tertiary Hybrid Behaviours

With the optimal ahead/behind pairing fixed to Predictive and FollowSightings, this phase tested the impact of adding a third specialized "search" behaviour for the no-memory state. Every available search algorithm was swept as the tertiary component.

Ahead/Behind Pair	No-Memory (Search) Behaviour	Average LOS%
FollowSightings + Predictive	**LevyBiasedStationSearchBehavior**	**71.97%**
FollowSightings + Predictive	SeparationBehaviour (ISR variant)	70.41%
FollowSightings + Predictive	ExpandingSpiralSearchBehavior	65.17%
FollowSightings + Predictive	LaneSweeperBehavior	60.79%
FollowSightings + Predictive	SeparationBehavior	59.59%

Table 88 Performance of Three-State Hybrid Configurations

As shown in **Table 3**, introducing a dedicated search behaviour provided another performance boost. The LevyBiasedStationSearchBehavior, with its intelligent blend of random exploration and a bias towards the home base proved to be the most effective strategy for target reacquisition, pushing the peak performance of the systematic pipeline to approximately 72%.

A Note on Heterogeneous Search Strategies: The RatioMode Experiment

Before finalizing the selection of a single search algorithm for the no-memory state a hypothesis was tested regarding the potential benefits of strategic diversity within the search phase itself. To this end a secondary state controller `RatioMode` was implemented. This controller was designed to manage a portfolio of search behaviours (e.g., `LevyBiasedStationSearchBehavior`, `LaneSweeperBehavior` and `ExpandingSpiralSearchBehavior`). When an agent entered the no-memory state, `RatioMode` would dynamically assign it one of these strategies, working to maintain a balanced, near-uniform distribution of active search patterns among its local neighbours.

The results of this experiment were conclusive: in all test configurations, the performance of a swarm using a single, optimized search behaviour consistently and significantly outperformed the heterogeneous mix managed by `RatioMode`. This finding suggests that while a diversity of strategies across different tactical contexts (ahead, behind, lost) is highly beneficial, a unified and coherent strategy within the critical reacquisition phase is more effective. A single, optimized search pattern allows the swarm to disperse and cover space in a predictable, systematic manner. Based on this outcome, the `RatioMode` approach was set aside, and the research proceeded with the selection of a single, best-in-class search algorithm for the final hybrid configuration.

Phase 4: Breakthrough configuration

While the systematic pipeline yielded a strong result, parallel exploratory design work revealed a critical tactical flaw in the existing behaviours. It was observed that even the best predictive algorithms tended to pursue the target from the sides, allowing for potential escape routes along its direct path. To solve this, two new, purpose-built behaviours were designed:

1. **GetInFrontBehavior (GIF):** A two-phase algorithm for the "ahead" state. It first moves to close the lateral gap with the target's velocity vector, then moves directly towards the target to form a "wall" and block its path.
2. **InverseSquareRepulsionBehavior (ISR):** A pure search behaviour for the "no-memory" state. It is functionally similar to the Separation Behaviour however for the repulsion vector it uses a strong, inverse-square repulsion force between agents and a weak attraction to their home bases, forcing the swarm to disperse rapidly and efficiently cover a wide area.

These new behaviours were integrated into the `HasMemory_DotProductMode` controller and subjected to the same rigorous, hyper-iterative optimization. The final, champion configuration was found to be:

- **Ahead State:** GetInFrontBehavior (GIF)
- **Behind State:** PredictiveInterceptMultiplierBehavior
- **No-Memory State:** InverseSquareRepulsionBehavior (ISR)

Ahead Behaviour	Behind Behaviour	No-Memory Behaviour	Average LOS%
GetInFrontBehavior	**PredictiveMultiplierBehavior**	**InverseSquareRepulsionBehavior**	**94.41%**
GetInFrontBehavior	PredictiveBehavior	InverseSquareRepulsionBehavior	88%

Table 99 Performance of Three-State Hybrid Configuration w/ Purpose Built Algorithms

This configuration (as shown in **Table.4**) achieved a final, stable average LOS Persistence Score of **94.41%.**

This result represents a massive 6× improvement over the initial baseline and a 31% relative improvement over the best outcome from the systematic search (Phase 1-3). It stands as a clear validation of the thesis hypothesis and demonstrates the profound effectiveness of combining a state-dependent architecture with purpose-built, specialized behaviours.

Qualitative Analysis of Champion Behaviours in Action

Quantitative scores are further illuminated by qualitative observation of the emergent swarm intelligence. The figures below illustrate the distinct and synergistic roles of the three champion behaviours during a simulation run.

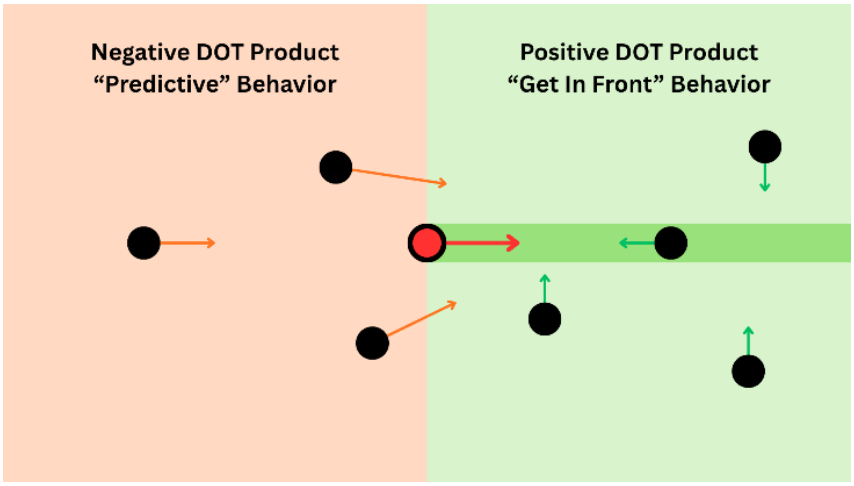

Figure 55 Has Memory Behaviour Illustrations

This diagram (**Figure.1**) illustrates the core logic of the state machine. When an agent (green) is ahead of the target (positive dot product), it activates the `GetInFrontBehavior`. When it is behind (negative dot product), it uses the `PredictiveInterceptMultiplierBehavior`.

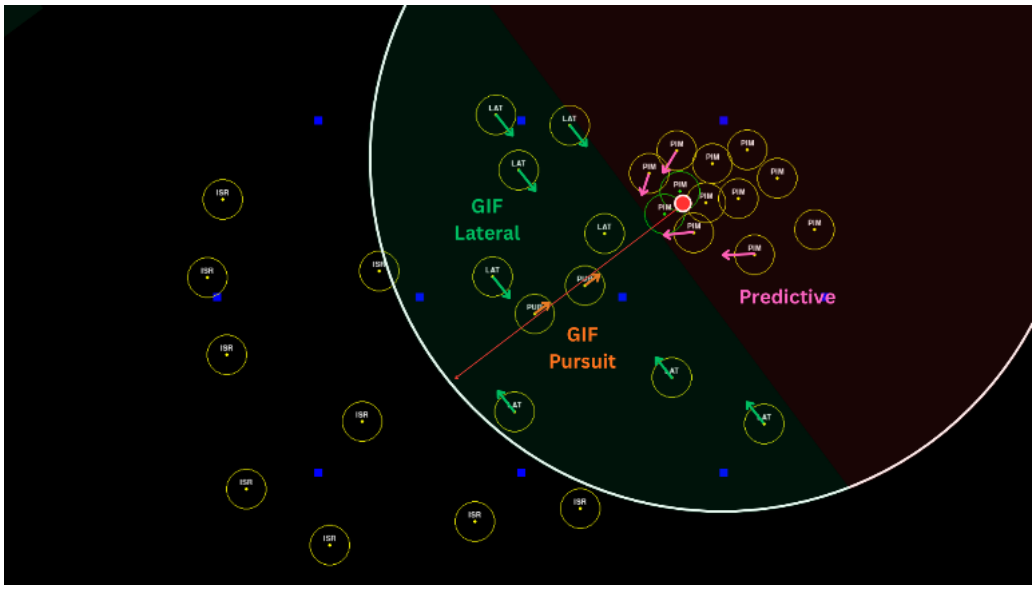

Figure 66 Behaviours in Action

This screenshot (**Figure.2**) from a live simulation visualizes the emergent coordination. Agents executing the GIF behaviour can be seen performing both the Lateral phase (moving to get in front) and the Pursuit phase (forming a wall). Agents further behind (labeled Predictive) are using the aggressive catch-up strategy to close the distance.

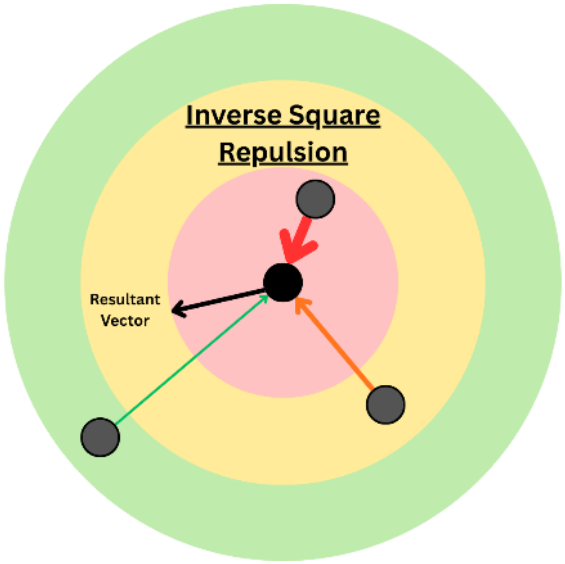

Figure 77 No Memory Behaviour Illustration

This conceptual diagram (**Figure.3**) shows the effect of the `InverseSquareRepulsionBehavior` (ISR). When memory is lost agents strongly repel from each other, forcing them to spread out and maximize the collective sensor footprint for efficient reacquisition rather than clustering at the last known position.

Together, these figures demonstrate the sophisticated, multi-part strategy that emerges from the simple, local rules of the state machine. The swarm dynamically creates a blocking formation, pursues aggressively, and disperses intelligently, leading to the robust and high-performance tracking observed in the results.

Summary of Experimental Findings

The iterative experimental pipeline, combining systematic evaluation with exploratory design, successfully guided the research from a low-performing baseline to a highly effective, state-dependent control system. The journey from a monolithic strategy to a specialized, multi-part hybrid demonstrates a clear and quantifiable progression of performance, validating the core thesis hypothesis. The table below provides a concluding summary of this performance evolution, highlighting the key insight gained at each major stage of the investigation.

Stage	Best Configuration	Key Insight	Peak LOS%
Baseline (S1)	Unoptimized SeparationBehavior	Monolithic strategies are insufficient.	14.60%
Phase 1	PredictiveMultiplierBehavior	Prediction is crucial but still limited.	55.63%
Phase 3	FollowSightings / Predictive / LevyBiased	A 3-state hybrid is superior.	71.97%
Final Solution	GIF / PredictiveMultiplier / ISR	Purpose-built behaviours achieve breakthrough.	94.41%

Table 1010: Summary of Performance Milestones

13 Discussion

The empirical results presented in the previous section demonstrate a clear and substantial progression culminating in a solution that achieves 94.41% LOS persistence a result that not only meets but exceeds the research aim. This section provides an interpretation of these findings, contextualizes their significance by revisiting the core behavioural sub-problems and discusses the broader implications of this work for the field of swarm robotics and its real-world applications.

Interpretation of Key Findings

The journey from a 14.6% baseline to a 94.41% characterize two overarching conclusions: the validation of the hybrid, state-dependent architectural hypothesis, and the critical role of purpose-driven design in achieving breakthrough performance.

The most significant finding of this research is the unequivocal validation that a hybrid, state-dependent control architecture is superior to any single, monolithic strategy for the fast-target pursuit problem. The performance jump observed between Phase 1 (peak of 55.63%) and Phase 2 (peak of 68.19%) is direct evidence of this. A monolithic behaviour, even a highly optimized one like `PredictiveMultiplierBehavior`, is forced to compromise as it cannot simply attempt to catch up from behind, intercept from the front and maintain spatial awareness.

The `HasMemory_DotProductMode` state machine resolves this conflict by delegating tactical responsibility. By decomposing the complex problem of "tracking" into simpler, context dependent sub-problems "intercepting," "pursuing," and "searching" it allows for the use of specialized algorithms that are optimized for a single job. This architectural choice proved to be the single most important factor in elevating the swarm's performance beyond the ceiling of a single unified behaviour.

The second key insight comes from the comparison between the results of the systematic pipeline (Phase 3) and the final exploratory solution (Phase 4). The systematic search, while rigorous, topped out at ~72% LOS. This represented a "local maximum" for the existing catalogue of behaviours. The performance leap to 94.41% was only possible through the purpose driven design of new algorithms to fill identified tactical gaps.

Qualitative analysis revealed that even predictive algorithms were inefficient at interception, as they tended to calculate intercept points perpendicular to the target's path. This led to agents converging from the sides leaving a crucial gap directly in front of the target. The `GetInFrontBehavior` (GIF) was designed specifically to close this gap by prioritizing lateral alignment before direct pursuit effectively forming a "wall." Similarly, the `InverseSquareRepulsionBehavior` (ISR) was engineered to solve the problem of post-memory clustering by enforcing a powerful, systematic dispersion. This demonstrates a crucial methodological insight: while automated optimization is a powerful tool for refinement, human-led, qualitative analysis and creative, targeted design are essential for overcoming the conceptual limitations of an existing solution set and achieving step changes in performance.

Addressing the Core Behavioural Sub-Problems

The success of the final GIF/PredictiveMultiplier/ISR configuration can be directly attributed to how its three components work in synergy to solve the distinct behavioural sub-problems identified in the Literature Review.

1. **Adaptive Pursuit:** This sub-problem is explicitly solved by the `HasMemory_DotProductMode` controller itself. The state machine provides the mechanism for dynamically switching between the specialized GIF interception strategy when ahead and the aggressive PredictiveMultiplier catch-up strategy when behind. This directly addresses the need for different tactics based on the agent's geometric relationship to the target.
2. **Target Reacquisition:** The ISR behaviour provides a robust and highly effective solution to this challenge. When LOS is lost and memory expires, the swarm's priority must shift from convergence to maximizing collective sensor coverage. The ISR's strong repulsive force prevents agents from clustering uselessly at the last known position a critical failure mode of simpler strategies. By forcing a rapid controlled dispersion it maximizes the probability that at least one agent will reacquire the target.
3. **Spatial Distribution:** The ISR also serves as the primary mechanism for maintaining effective spatial distribution, but only during the critical "lost" phase. During active pursuit, crowd control is managed by the hard separation components within the GIF and PredictiveMultiplier behaviours. The system thus uses two different modes of spatial management: a tight, coordinated formation during engagement, and a rapid, dispersive formation during search.

4. **Information Propagation**: All three states of the champion configuration leverage the k-nearest neighbour communication network to share sighting information. This is most critical in the transition from a "lost" state to a "pursuit" state. As soon as a single agent executing the ISR search pattern reacquires the target its updated sighting information is propagated to its local neighbours. This triggers a cascading state change, causing nearby searching agents to switch back to pursuit behaviours, allowing the swarm to rapidly re-converge on the target.

While conducted in a 2D, obstacle-free simulation, the parameters and the problem statement were explicitly chosen to reflect real-world operational constraints strengthening the relevance of these findings.

The 2:1 speed deficit between the target and the agents represents a realistic and challenging pursuit scenario. The achievement of 94.41% LOS persistence demonstrates that a swarm of relatively low-cost, "slower" assets can be a highly effective and resource-efficient substitute for a single expensive high-speed asset. This has significant implications for applications in law enforcement, security and surveillance where deploying a swarm could offer greater coverage, robustness, and cost-effectiveness.

A key motivation for this research was to explore the under-researched area of deployment strategies. The success of our final configuration demonstrates that if a swarm's intrinsic adaptive behaviours are sufficiently robust particularly its reacquisition strategy the need for complex base logistics can be minimized or even eliminated for missions of a certain duration. This implies that for many practical applications, the most critical factor is not managing a reserve of assets, but ensuring the initially deployed cohort is as intelligent and cooperative as possible. This finding helps to focus future research and development on behavioural design over complex logistical frameworks.

This research contributes a valuable methodological blueprint for swarm engineering. The 10-step *Iterative Design and Optimization Methodology* combines the automated data-driven parameter tuning with the creative potential of human in the loop analysis and design. The success of this dual approach suggests that the most effective path to developing complex, high-performance robotic systems lies in a hybrid methodology where systematic optimization is used to refine ideas and qualitative observation is used to generate new ones.

In summary, the discussion of the results reveals more than just a high performing algorithm. It validates a specific architectural paradigm (state dependent control), highlights a powerful design methodology (purpose driven iteration) and provides a concrete data backed solution that directly addresses the core research question. The findings have significant implications, suggesting that behaviourally adaptive, deployment-centric swarms represent a robust and efficient solution to the challenging fast-target pursuit problem.

Limitations and Engineering Trade-offs

While the final behavioural configuration demonstrates high efficacy, its development and performance were shaped by several deliberate engineering trade-offs and necessary design limitations. Acknowledging these is crucial for contextualizing the results and defining avenues for future work.

A primary trade-off considered was between strategic diversity and optimized specialization in the search phase. The `RatioMode` experiment was designed to test the hypothesis that a heterogeneous mix of search behaviours (Lévy, Spiral, Lane-sweeper) would provide more robust area coverage than a single method. The results, however, conclusively favoured a single, highly optimized strategy (ISR). This indicates a key trade-off: while a diverse portfolio of strategies offers flexibility, the performance gains from a unified, coherent, and globally optimized search pattern are superior for this specific problem. The swarm benefits more from all agents executing a predictable, mutually understood dispersal pattern than from a mix of uncoordinated, albeit individually effective, strategies.

Another fundamental trade-off lies in the complexity versus performance of the control architecture. The initial monolithic behaviours were simpler to implement and tune but were performance limited by their "one size fits all" nature. The move to the `HasMemory_DotProductMode` state machine introduced significant architectural complexity, requiring the design, tuning, and integration of multiple sub behaviours. This trade off, accepting higher design complexity for a substantial gain in performance, is central to the project's success and validates the initial research hypothesis.

The findings must also be understood within the context of the simulation's limitations. The design of the simulator intentionally prioritized behavioural dynamics over environmental realism, leading to several key abstractions:

1. **Environment Fidelity:** The simulation operates in a 2D, obstacle-free environment. This was a necessary simplification to isolate and analyse the core effects of agent behaviours without the confounding variable of path-planning or collision avoidance with static objects. The transferability of these results to a complex, 3D, or obstacle-rich environment remains an open question.
2. **Perfect Communication and Sensing:** The current model assumes a perfect, noise-free, and zero latency communication network for the k-nearest neighbours. Similarly, agent sensing is binary and perfect within the defined Agent sight radius. In a real-world deployment, communication packet loss, latency, and sensor noise would degrade situational awareness and likely impact the swarm's coordination and LOS persistence.
3. **No Physical Constraints:** The simulation abstracts away physical constraints such as agent battery life (energy), differing sensor payloads, or potential hardware failures. The current strategy optimizes for time on target not energy efficiency which would be a critical factor in a real-world drone deployment.

These limitations do not invalidate the results rather, they define the boundaries of the current claims and provide a clear and fertile ground for future research, which is further explored in the final chapter of this thesis.

14 CONCLUSION & FUTURE WORK

This thesis set out to address a key challenge in swarm robotics: the persistent tracking of a target possessing significantly superior speed. This final chapter synthesizes the outcomes of this research beginning with a summary of the key findings from the iterative design and evaluation process. It then provides concrete recommendations for the immediate validation and improvement of the current solution, before concluding with a discussion of promising avenues for future work that could extend these findings into more complex and realistic domains.

Summary of Findings

1. Validation of the Hybrid Hypothesis: The research unequivocally demonstrated that a hybrid, state dependent control architecture is superior to any monolithic behaviour. The introduction of the `HasMemory_DotProductMode` state machine, which delegates tasks based on the agent's tactical context (ahead of, behind, or without knowledge of the target), marked the most significant architectural advance, elevating performance from ~56% to ~72% LOS. This confirmed that decomposing the complex tracking problem into specialized sub-problems is a highly effective strategy.
2. The Breakthrough of Purpose-Driven Design: While systematic optimization of existing behaviours provided substantial gains, the final performance breakthrough was achieved through a parallel process of qualitative analysis and purpose-driven design. The creation of the `GetInFrontBehavior` (GIF) to solve the specific tactical problem of interception, and the `InverseSquareRepulsionBehavior` (ISR) to solve the problem of post-memory clustering, resulted in a final, champion configuration.
3. Achievement of the Research Aim: The final three-part hybrid configuration, GIF (Ahead) / PredictiveMultiplier (Behind) / ISR (No-Memory) achieved a stable, average LOS persistence of 94.41% across all test paths. This result represents a 6-fold improvement over the initial baseline and a 31% relative improvement over the best systematically derived solution, thereby meeting and exceeding the primary research objective.

Recommendations for Improvement

1. **Generalization and Overfitting Tests:** The final parameters were optimized against a fixed suite of four target paths. To ensure the solution is not overfitted to these specific scenarios, a crucial

next step is to test the champion configuration without modification against a new unseen set of validation paths. These paths should incorporate more complex and non-deterministic patterns, such as sharp turns, stop-and-go manoeuvres, and randomized waypoints to rigorously assess the generalizability of the strategy.

2. **Testing Against Evasive Targets**: A key simplification in this research was the use of non-evasive targets. This was a necessary and deliberate choice to isolate the swarm's tracking and reacquisition capabilities without the confounding variable of a reactive target. The next logical extension is to introduce simple evasive logic to the target (e.g., moving away from the nearest agent or the swarm's centroid). This would test the robustness of the GIF "walling" strategy and the reacquisition speed of the ISR search pattern against a more challenging, interactive opponent.
3. **Scalability and Resource Efficiency Analysis:** The current results were achieved with a fixed swarm size of 30 agents. Further analysis is required to determine the solution's scalability. This should involve two lines of inquiry:
    - **Agent Number Scaling:** Systematically varying the number of agents (e.g., from 10 to 50) to determine the minimum number required to consistently maintain >90% LOS persistence. This provides a direct measure of the solution's resource efficiency.
    - **Speed Ratio Impact:** Exploring the performance impact of changing the target to agent speed ratio (e.g., 1.5:1, 3:1). This will help to define the effective operational limits of the strategy and its applicability to targets of varying speeds.

Future Work

1. **Increased Environmental and Physical Complexity:** To enhance realism, the simulation could be augmented with several key features:
    - **Environmental Constraints:** The introduction of static obstacles (buildings) and dynamic no-fly zones would test the agents' ability to maintain coordination while navigating a constrained environment necessitating the integration of path planning heuristic algorithms (e.g., A*) into the behavioural logic.
    - **Communication Modelling:** The current assumption of a perfect communication network could be replaced with more realistic models that include latency, packet loss, and range limitations. This would test the robustness of the information propagation mechanism, which is critical to the swarm's cohesive function.
    - **Agent Energy Models:** Incorporating a battery life model for each agent would introduce a new and critical optimization challenge: maximizing LOS persistence per unit of energy consumed. This could lead to the development of energy-aware behaviours, where agents might reduce speed or enter a low-power "loitering" state when not actively pursuing.
2. **Transition to Hardware:** The ultimate validation of this research would be the transfer of the champion behavioural configuration to a physical swarm. A team of small, agile quadcopters in a controlled indoor environment would be an ideal platform for this. This would test the solution's real-world efficacy and surface challenges related to state estimation, control latency, sensor noise and even physical limitations that are not considered in simulation testing.
3. **Comparative Analysis with Machine Learning Approaches:** The heuristic parameter-driven approach of this thesis offers high explainability and tunability. A compelling avenue for future research would be to use the current simulation environment to train a Multi-Agent Reinforcement Learning (MARL) agent on the exact same problem. A comparative analysis between the engineered GIF/PRED/ISR solution and the learned MARL policy would provide valuable insights into the trade-offs between designed and emergent strategies in terms of performance, adaptability, sample efficiency and the "black box" problem of explainability [19].

15 REFERENCES

[1] L. E. Parker, "Distributed algorithms for multi-robot observation of multiple moving targets," Auton. Robot., vol. 12, pp. 231–255, May 2002, doi: 10.1023/A:1015256330750.

[2] International Aeronautical and Maritime Search and Rescue Manual (IAMSAR), Volume III: Mobile Facilities, 10th ed. London, U.K. and Montréal, Canada: International Maritime Organization and International Civil Aviation Organization, 2016. ISBN 978-92-9258-059-9.

[3] M. M. Shahzad et al., "A review of swarm robotics in a nutshell," Drones, vol. 7, no. 4, Apr. 2023, Art. no. 269, doi: 10.3390/drones7040269.

[4] H. L. Kwa, J. L. Kit, and R. Bouffanais, "Tracking multiple fast targets with swarms: Interplay between social interaction and agent memory," in Proc. 2021 Conf. Artif. Life, Jul. 2021, pp. 484–493, doi: 10.1162/isal_a_00376.

[5] M. Senanayake, I. Senthooran, J. C. Barca, H. Chung, J. Kamruzzaman, and M. Murshed, "Search and tracking algorithms for swarms of robots: A survey," Robot. Auton. Syst., vol. 75, pp. 422–434, Jan. 2016, doi: 10.1016/j.robot.2015.08.010.

[6] M. Kegeleirs and M. Birattari, "Towards applied swarm robotics: current limitations and enablers," Frontiers in Robotics and AI, vol. 12, Art. no. 1607978, Jun. 2025, doi: 10.3389/frobt.2025.1607978.

[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proc. Int. Conf. Neural Netw. (ICNN'95), Perth, WA, Australia, 1995, vol. 4, pp. 1942–1948, doi: 10.1109/ICNN.1995.48896488968.

[8] M. G. M. Hamami and Z. H. Ismail, “A systematic review on particle swarm optimization towards target search in the swarm robotics domain,” Archives of Computational Methods in Engineering, vol. 30, no. 5, pp. 4051–4089, Oct. 2022, doi: 10.1007/s11831-022-09819-3.

[9] T. Muslimov, “Particle Swarm Optimization for Target Encirclement by a UAV Formation,” in Proc. 15th Int. Conf. Intelligent Systems (INTELS’22), Moscow, Russia, Dec. 14–16, 2022, Eng. Proc., vol. 33, p. 15, Jun. 2023. doi: 10.3390/engproc2023033015

[10] L. E. Parker and B. A. Emmons, "Cooperative multi-robot observation of multiple moving targets," in Proc. IEEE/RSJ Int. Conf. Grenoble, France, Sep. 1997, pp. 925–932, doi: 10.1109/IROS.1997.655115.

[11] M. Kouzeghar, Y. Song, M. Meghjani, and R. Bouffanais, "Multi-Target Pursuit by a Decentralized Heterogeneous UAV Swarm using Deep Multi-Agent Reinforcement Learning," in Proc. 2023 IEEE Int. Conf. Robot. Autom. (ICRA), May 29–June 2, 2023, London, UK, pp. 3289–3295, doi: 10.1109/ICRA48891.2023.10160919.

[12] G. Singh, D. Lofaro, and D. Sofge, "Pursuit-evasion with decentralized robotic swarm in continuous state space and action space via deep reinforcement learning," in Proc. Int. Conf. Agents Artif. Intell. (ICAART), Valletta, Malta, Feb. 2020, pp. 226–233, doi: 10.5220/0008971502260233.

[13] V. Gabler and D. Wollherr, "Decentralized multi-agent reinforcement learning based on best-response policies," Front. Robot. AI, vol. 11, Apr. 2024, Art. no. 1229026, doi: 10.3389/frobt.2024.1229026.

[14] M. Mersha, M. Ghebreab, and R. Asgedom, “Interpreting Black-Box Models: A Review on Explainable Artificial Intelligence,” Cognitive Computation, vol. 15, pp. 2219–2244, 2023. [Online]. Available: https://doi.org/10.1007/s12559-023-10179-8

[15] D. Marthaler, A. L. Bertozzi, and I. B. Schwartz, “Levy searches based on a priori information: The Biased Levy Walk,” Tech. Rep., Dept. of Mathematics, Univ. of California, Los Angeles, and Plasma Physics Div., US Naval Research Laboratory, Washington, DC, 2004.

[16] Y. Katada and K. Ohkura, "Swarm robots using Lévy walk with concession in targets exploration," Artificial Life and Robotics, vol. 28, pp. 652–660, Oct. 2023, doi:10.1007/s10015-023-00900-z.

[17] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," IEEE Trans. Robot., vol. 21, no. 3, pp. 376–386, Jun. 2005, doi: 10.1109/TRO.2004.839232.

[18] A. K. Larsen, K. I. Kilic, M. B. Ladefoged, and I. Sung, "A novel path-finding approach for maritime search-and-rescue missions incorporating dynamic probability of a target location," Engineering Optimization, published online Mar. 6, 2025, doi: 10.1080/0305215X.2024.2446590.

[19] O. Zedadra, A. Guerrieri, and H. Seridi, "LFA: A Lévy Walk and Firefly-Based Search Algorithm: Application to Multi-Target Search and Multi-Robot Foraging," Big Data Cogn. Comput., vol. 6, art. no. 22, Feb. 2022, doi: 10.3390/bdcc6010022.

16 APPENDICES

Appendix A – Simulation Code Repository

The complete simulation source code, including configuration files, behavioural modules, and experimental scripts used in this research, is available at the following GitHub repository:

https://github.com/gabefromvbucks/swarm_tracking_simulation_V2.git

This repository contains all relevant Python files, parameter definitions, and visualisation tools necessary to reproduce the experiments and results presented in this thesis.