# Formal Modeling and Probabilistic Verification of a Decentralized Drone Swarm Coordination

Garima Dhakal

Technische Universität Dresden

**Abstract**

As autonomous drone swarms are increasingly used for safety-critical missions, formal methods are essential to guarantee their operational safety and reliability. This report presents the formal modeling and quantitative verification of a decentralized coordination protocol for drones navigating a cluttered, grid-based environment. The system is modeled as a Markov Decision Process (MDP) to capture both the non-deterministic directional choices of the drones and the probabilistic outcomes of their movements. A key simplifying assumption for model tractability is that the collision avoidance mechanism checks only against the current static positions of other drones. Using the PRISM model checker, we perform a quantitative analysis of mission success and systemic failure modes, such as the probability of all drones crashing simultaneously. The results reveal a key trade-off between movement predictability and system resilience in worst-case scenarios. Finally, we demonstrate the challenge of state-space explosion, which limits the scalability of this formal approach and establishes a critical direction for future research.

## 1 Introduction

Drone swarms are rapidly emerging as a transformative technology with broad applications in fields such as cooperative surveillance, precision agriculture, logistics, and disaster response. By leveraging the collective capabilities of multiple simple agents, swarms offer significant advantages in scalability, robustness, and cost-effectiveness compared to single, monolithic robotic systems. The emergent behavior of a swarm allows it to perform complex tasks that would be difficult or impossible for an individual unit to accomplish.

Figure 1: A synchronized movement of a drone.

However, the effective coordination of these swarms presents a formidable challenge. Ensuring that a group of autonomous agents can navigate dynamic and cluttered environments to achieve a common objective requires sophisticated solutions to several interconnected problems, including formation control, trajectory

planning, and, most critically, collision avoidance. The complexity of this coordination task increases non-linearly with the number of agents, posing a significant barrier to the scalability of swarm operations [5]. The core problem in swarm robotics is designing simple, local interaction rules for individual agents that reliably produce a desired, emergent global behavior. This project investigates this fundamental link by designing a decentralized protocol based on local agent decisions and using formal verification to rigorously check whether these local rules achieve the global objective of safe and successful mission completion.

For applications where safety and reliability are paramount, traditional validation methods like physical testing and simulation are often insufficient. Simulation can explore a large number of scenarios but cannot provide exhaustive guarantees, while physical testing is expensive and often impractical for covering all edge cases. Formal methods, in contrast, offer a suite of mathematically-grounded techniques for the rigorous analysis of system behavior. Among these, probabilistic model checking has become an essential tool for analyzing systems that operate in the presence of uncertainty, such as communication failures or unpredictable environmental interactions [1, 4]. It provides a means to compute quantitative, provable guarantees about a system's adherence to specified properties.

# 2 Preliminaries

This section provides the necessary background for the formal modeling and verification of the drone swarm protocol. We first discuss common coordination paradigms in multi-agent systems to provide domain context. We then introduce the mathematical formalism of Markov Decision Processes (MDPs), which underpins our system model. The section concludes with an overview of the verification tools, PRISM and ProFeat, used in this study.

## 2.1 Drone Swarm Coordination Paradigms

Control architectures for drone swarms are typically categorized as either centralized or decentralized. In a **centralized** architecture, a single ground station or a designated leader agent is responsible for processing all information, making decisions, and issuing commands to the other drones. This approach simplifies decision-making and can ensure consistent behavior across the swarm. However, it suffers from significant drawbacks, including a single point of failure—if the central controller is lost, the entire swarm may fail—and communication bottlenecks that limit scalability.

In contrast, a **decentralized** architecture distributes decision-making authority among the individual agents in the swarm. Each drone makes its own decisions based on local information, such as its own state and the states of its immediate neighbors. This approach is inherently more robust, fault-tolerant, and scalable, as the system can continue to function even if some agents fail. The protocol developed in this project is decentralized, with each drone independently choosing its next intended move [5].

While the protocol itself is decentralized, the verification method used to analyze it is fundamentally centralized. Probabilistic model checking, as implemented in PRISM, requires constructing a global model that encompasses all possible states of the entire multi-agent system. The model checker then performs an exhaustive search of this global state space to verify properties [1]. This conceptual tension between a decentralized system design and a centralized analysis technique is a key theme in the verification of multi-agent systems. It is precisely this need for a global view during verification that leads to the primary challenge of this work: the state-space explosion problem.

## 2.2 Formalizing Stochastic Systems: Markov Decision Processes (MDPs)

To formally reason about a system that involves both agent choices and probabilistic uncertainty, we use the framework of Markov Decision Processes (MDPs). An MDP is a mathematical model for sequential decision-making in a stochastic environment, making it an ideal choice for capturing the dynamics of our drone swarm [5, 1]. At each step in an MDP, an agent in a given state chooses an available action. The environment then responds by transitioning to a new state with a certain probability, and the agent receives a corresponding reward or cost.

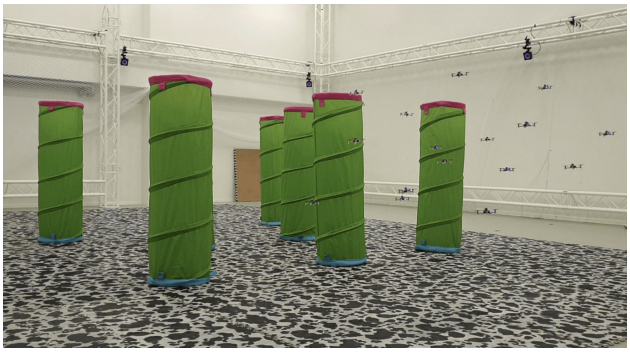


Figure 2: A physical testbed environment for drone swarm navigation with obstacles.

**Definition.** ***(Markov decision process)*** *A* Markov decision process *(MDP) is a tuple* $M = (S, s_{\text{init}}, A, P, R)$*, where:*

- $S$ *is a finite set of states. In our context, a state is a complete snapshot of the system, capturing the position and status of every drone.*
- $s_{\text{init}} \in S$ *is the initial state of the system.*
- $A$ *is a finite set of actions. Actions represent the choices available to the system's agents.*
- $P : S \times A \times S \to [0, 1]$ *is the probabilistic transition function, where* $P(s, a, s')$ *is the probability of transitioning from state* $s$ *to state* $s'$ *after taking action* $a$*. For any given state-action pair* $(s, a)$*, the probabilities of transitioning to all possible next states* $s'$ *must sum to 1.*
- $R : S \times A \to \mathbb{R}$ *is a reward (or cost) function that assigns a value to taking an action in a state.*

A *policy* (or scheduler) is a function that resolves the choices of actions. For any given state, a policy specifies which action to take. The goal of verification is often to find optimal or worst-case policies with respect to some performance objective, such as maximizing the probability of reaching a goal or minimizing the expected cost. A scheduling strategy is the concrete way in which a scheduler resolves non-deterministic choices during verification as it determines, which action a drone will attempt when multiple directions are possible.

It is crucial to distinguish between non-determinism and probabilism within the MDP framework, as they represent two different types of uncertainty.

- **Non-determinism** represents the choices that are available to an agent or a scheduler. Imagine an agent at a crossroads in a maze. It has a choice between two actions: `take_left_path` or `take_right_path`. The model does not assign a probability to which path the agent will choose; this decision is resolved by a policy. An optimal policy might choose the path that leads to the exit faster, while an adversarial policy might choose the path leading to a trap.
- **Probabilism** represents the inherent, stochastic uncertainty in the outcome of an action once it has been chosen. For example, after the agent's policy selects the action `take_left_path`, the path itself might be unreliable. There could be an 80% probability of successfully reaching the next junction, but a 20% probability of encountering an unexpected obstacle that forces the agent back to the same crossroads. This probabilistic outcome is an intrinsic property of the environment and is independent of the policy's choice.

## 2.3 Modeling and Verification Tools

This sub-section introduces the primary software tools used for modeling and verifying the system. We use PRISM as the core engine for probabilistic model checking and ProFeat to enable a feature-oriented and modular modeling approach.

### 2.3.1 PRISM for Quantitative Verification

PRISM is a powerful, open-source software tool for the formal modeling and analysis of systems that exhibit probabilistic behavior. It supports a variety of probabilistic models, including Discrete-Time Markov Chains (DTMCs), Continuous-Time Markov Chains (CTMCs), and, most relevant to this work, Markov Decision Processes (MDPs) [4].

The core functionality of PRISM is to automatically verify whether a formal model of a system satisfies a given property. These properties are specified using extensions of temporal logic, such as Probabilistic Computation Tree Logic (PCTL). PCTL allows for the expression of complex quantitative queries. For instance, the query $P_{max}$ =?[$F$ "goal" ] asks for the maximum probability, across all possible policies, of eventually reaching a state labeled "goal" [5]. PRISM analyzes the model exhaustively to compute exact numerical answers to such queries, providing rigorous guarantees that are unattainable through simulation alone [1].

### 2.3.2 ProFeat for Feature-Oriented Modeling

Feature-Oriented Software Development (FOSD) is a paradigm for engineering families of related software systems, known as software product lines [3]. The idea is to encapsulate distinct functionalities into modular units called "features," which can be combined in various ways to generate different system variants [2].

ProFeat is a tool developed at TU Dresden that extends the PRISM modeling language with concepts from FOSD. It allows developers to describe families of probabilistic systems in a modular and scalable way. By encapsulating different system components or behaviors into separate feature modules, ProFeat promotes a clean and maintainable design. This feature-based structure makes it easier to manage model variants, for example, by configuring a system with different numbers of interacting agents or alternative behavioral protocols.

# 3 System Model and Protocol Definition

This section provides a formal and detailed description of the drone swarm coordination system. We define the environment, the state and behavior of the agents, and the system dynamics. The model is implemented in the PRISM language and structured using the feature-oriented capabilities of ProFeat.

## 3.1 Environment and Scenario Setup

The operational environment for the drone swarm is a discrete, two-dimensional grid. The key parameters are defined as constants in the PRISM model as follows:

- **Grid:** The environment is a grid $G = \{0, ..., N-1\} \times \{0, ..., M-1\}$. For the primary case study, we use a $5 \times 5$ grid ($N = M = 5$).

```
const int GRID_SIZE = 5;
const int GRID_MAX  = GRID_SIZE - 1;

```

Listing 1: Grid size definition in PRISM.

- **Drones:** The system comprises a set of $k$ drones, $D = \{d_0, ..., d_{k-1}\}$. The main analysis focuses on a swarm of $k = 3$ drones.

```
const int drone_no = 3;

```

Listing 2: Defining the number of drones.

- **Start and Goal States:** Each drone $d_i \in D$ is assigned a unique starting position $(x_i, y_i)$ and a unique target goal position $(gx_i, gy_i)$.

```
const int INIT_X[] = {0, 0, 4};
const int INIT_Y[] = {0, 3, 0};
const int GOAL_X[] = {4, 4, 0};
const int GOAL_Y[] = {4, 0, 4};

```

Listing 3: Initial and goal coordinates for the drones.

- **Obstacles:** The environment contains a set of static, impassable obstacles, defined by their coordinates $O \subset G$.

```
formula obstacle(x,y) = (x=2 & y=1) | (x=1 & y=3) | (x=3 & y=4);

```

Listing 4: Obstacle locations defined as a formula.

The overall objective is for every drone to navigate from its start position to its goal position while avoiding all hazards. The specific parameters are summarized in Table 1.

Table 1: System Parameters for the Primary Case Study

| **Parameter** | **Value** |
|---|---|
| Grid Size | $5 \times 5$ |
| Number of Drones | 3 |
| Drone 0 ($d_0$) Start $\rightarrow$ Goal | $(0,0) \rightarrow (4,4)$ |
| Drone 1 ($d_1$) Start $\rightarrow$ Goal | $(0,3) \rightarrow (4,0)$ |
| Drone 2 ($d_2$) Start $\rightarrow$ Goal | $(4,0) \rightarrow (0,4)$ |
| Obstacle Locations | $\{(2,1),(1,3),(3,4)\}$ |

## 3.2 Modular Design with ProFeat

To manage the complexity of the multi-agent system and facilitate scalability, the model is designed in a modular, feature-oriented fashion using ProFeat. Each drone's core logic, its decision-making process for selecting a direction, and the system-wide reward structures are encapsulated in separate features. This separation of concerns promotes a clean and maintainable design. The high-level feature composition is shown in Listing 5.

```
root feature
    all of drone[drone_no], Rewards, moveDirection[drone_no];
endfeature

feature drone
    modules drone_implementation(id);
endfeature

feature moveDirection
    modules moveDirection_impl(id);
endfeature

feature Rewards
    rewards "steps_to_goal"
        [move] true : 1;
    endrewards
endfeature
```

Listing 5: ProFeat code defining the feature-based structure of the model.

### 3.3 Agent State and Behavior Model

The system's behavior is captured by each agent's state and the rules governing its transitions. This is where the theoretical distinction between non-determinism and probabilism is realized in the model.

- **State Variables:** A global state in the MDP is a composite of the individual states of all drones. The state of a single drone $d_i$ is a tuple $(x_i, y_i, st_i)$, where $(x_i, y_i)$ are its coordinates and $st_i$ is its status from {`active, goal, crashed`}.
- **Actions and Transitions:** At each step, an active drone first non-deterministically chooses a direction, and then its movement is resolved probabilistically. This two-phase process is shown in Listing 6. The non-deterministic choice is modeled by distinct labeled actions (`select_north`, etc.), which a policy resolves. The subsequent probabilistic movement is modeled as a single command (`move`) with multiple potential outcomes based on the chosen direction.

`drone_id`.

```
// Phase 1: Non-deterministic choice resolved by a policy/scheduler
[select_north[drone_id]] direction=0 & !drone_crash -> (direction'=1);
[select_south[drone_id]] direction=0 & !drone_crash -> (direction'=2);
[select_east[drone_id]]  direction=0 & !drone_crash -> (direction'=3);
[select_west[drone_id]]  direction=0 & !drone_crash -> (direction'=4);

// Phase 2: Probabilistic outcome of the 'move' command for a southerly choice
[move] !drone_crash & direction=2 ->
    P_STRAIGHT  : (y_loc'=y_loc-1) & (x_loc'=x_loc) & (direction'=0) +
    P_DEVIATION : (y_loc'=y_loc-1) & (x_loc'=x_loc+1) & (direction'=0) +
    P_DEVIATION : (y_loc'=y_loc-1) & (x_loc'=x_loc-1) & (direction'=0);
```

Listing 6: PRISM implementation of non-deterministic choice and probabilistic outcome for a single drone

The probabilistic movement model for a drone intending to move South is visualized in Figure 3.

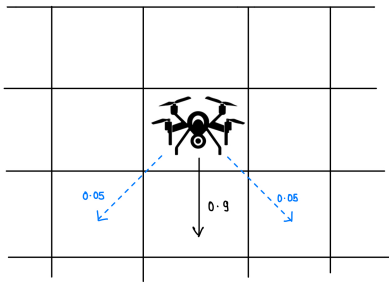



Figure 3: Probabilistic movement model for a drone intending to move South.

### 3.4 System Dynamics and Failure Conditions

The system evolves in synchronous rounds. A drone's mission fails if it enters a crash state, which is an absorbing state.

- **Crash Conditions:** A drone $d_i$ transitions to the `crashed` status if its next intended position $(x_i', y_i')$ satisfies any of the following:
  1. **Boundary Collision:** The target is outside the grid boundaries.
  2. **Obstacle Collision:** The target coincides with a static obstacle.
  3. **Inter-Drone Collision:** The target position of drone $d_i$ is a cell that is **already occupied by the current position** of another drone $d_j$.

These conditions are implemented as guards on the PRISM commands that govern movement. A move is only successful if it avoids all such hazards, as shown in the example guard in Listing 7. If the guard for a successful move is false, a separate command transitions the drone to the 'crashed' state.

- **Goal State:** If a drone $d_i$ reaches its goal coordinates, its status becomes `goal`. This is also an absorbing state.

```
// Guard for a successful move North for drone_id
[move] !drone_crash & direction=1 & is_in_grid(x_loc,y_loc+1) &
       !obstacle(x_loc, y_loc+1) & !is_drone_crash(drone_id,x_loc,y_loc+1) -> ...
```

Listing 7: Example of a command guard used to check for failure conditions before a successful move.

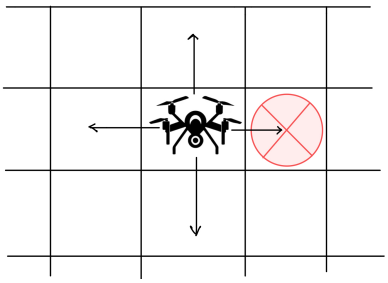
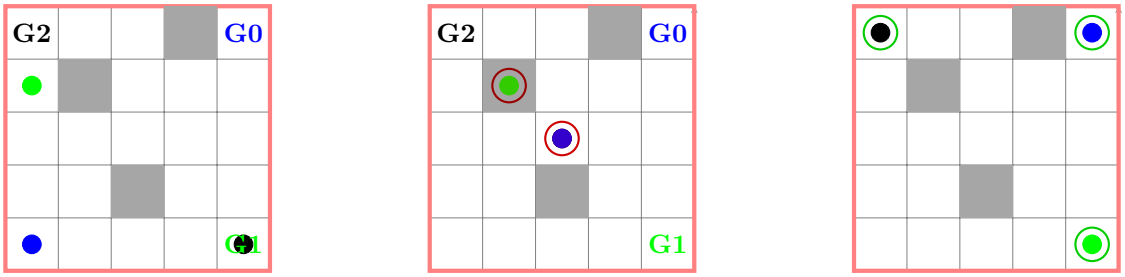


Figure 4: Illustration of a drone navigating a grid with a potential obstacle collision.

### 3.5 Illustrative Scenarios

To better understand the system dynamics, this subsection visualizes potential outcomes. Figure 5(a) shows the starting configuration. An adversarial policy can exploit non-determinism to force choices that lead to a high probability of collision. For example, it could direct two drones towards each other. A possible outcome of such a worst-case policy is shown in Figure 5(b), where multiple drones have crashed. Conversely, an optimal policy will choose actions that maximize the likelihood of all drones safely reaching their goals, as depicted in Figure 5(c).




(a) Initial state (b) Worst-case outcome (c) Best-case outcome

Figure 5: Illustration of the initial state, a potential worst-case outcome (drones crashes), and the desired best-case outcome (all goals reached).

## 4 Experimental Verification and Analysis

This section presents the results obtained from the probabilistic model checking of the drone swarm system.

## 4.1 Properties for Verification

In model checking, a property is a formal specification of a desired or undesired behavior of the system. These properties are typically expressed in a temporal logic, such as PCTL for probabilistic systems. By verifying these properties against the model, we can obtain quantitative answers to critical questions about the system's performance and safety. For an MDP, we can distinguish between best-case and worst-case scenarios by querying for the maximum ($P_{max}$) and minimum ($P_{min}$) probability of a property holding, over all possible policies. We focus on overall mission success as well as systemic crash properties, where the entire swarm fails simultaneously, as defined in Table 2.

Table 2: Verified System Properties in PCTL

| **Property Description** | **PCTL Query** |
|---|---|
| Max/Min probability of Drone i reaching its goal | $P_{max/min}$ =?[$F$ "goal_i" ] |
| Max/Min probability of all drones reaching goals | $P_{max/min}$ =?[$F$ "all_goals" ] |
| Max/Min probability of all drones being in a collision state | $P_{max/min}$ =?[$F$ "all_collisions" ] |
| Max/Min probability of all drones hitting an obstacle simultaneously | $P_{max/min}$ =?[$F$ "all_obstacle_hit" ] |
| Max/Min probability of all drones crashing for any reason | $P_{max/min}$ =?[$F$ "all_crash" ] |

## 4.2 Analysis of Goal Reachability and Safety

Our experiments reveal a direct and critical relationship between a drone's movement predictability ($P_{\text{STRAIGHT}}$) and the overall mission success. As shown in Figures 6 and 7, the maximum probability of successfully reaching the goals degrades significantly as the movement becomes more random.

With a nearly deterministic model ($P_{\text{STRAIGHT}} = 0.99$), an optimal scheduling policy can achieve a 90.7% probability of all three drones reaching their goals. However, as unpredictability increases, this success rate plummets. When $P_{\text{STRAIGHT}}$ is reduced to 0.85, the maximum probability of success ($P_{max}$("all_goals")) falls to just 13%, and further to a mere 3% when $P_{\text{STRAIGHT}} = 0.75$.

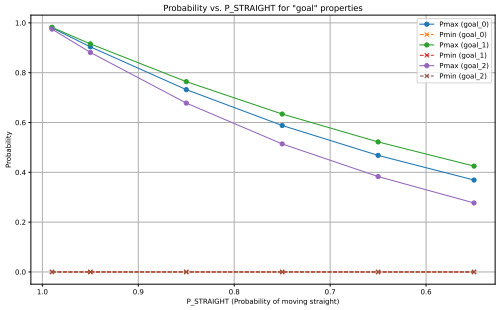



Figure 6: Probability of individual drones reaching their goals.

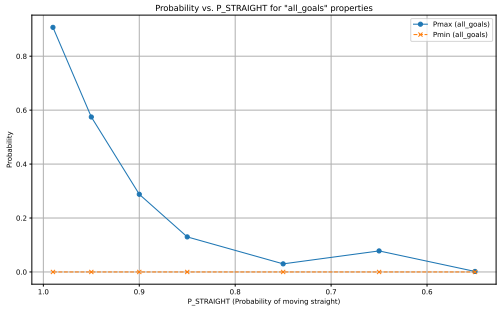



Figure 7: Probability of all drones simultaneously reaching their goals.

Conversely, the probability of failure modes like collisions rises dramatically with increased randomness (Figure 9). Under a worst-case scheduler actively trying to cause a collision, the probability ($P_{max}$) is always 1.0. A more insightful metric is the minimum probability of collision ($P_{min}$), which represents the chance of a collision occurring even when the scheduler is optimally trying to prevent it. At $P_{\text{STRAIGHT}} = 0.95$, this minimum collision probability is already 42.5%. It climbs to 86.9% at $P_{\text{STRAIGHT}} = 0.85$, indicating that as movement becomes less predictable, collisions become almost unavoidable regardless of the scheduling policy.

This wide gap between $P_{max}$ and $P_{min}$ at high values of $P_{\text{STRAIGHT}}$ highlights the power of the scheduler in a predictable system. As randomness increases, this gap narrows, and the system's outcome becomes inherently probabilistic. This reveals a crucial trade-off: **increasing randomness severely reduces the potential for optimal success, but it also makes the system's behavior less dependent on a perfect scheduling strategy.**

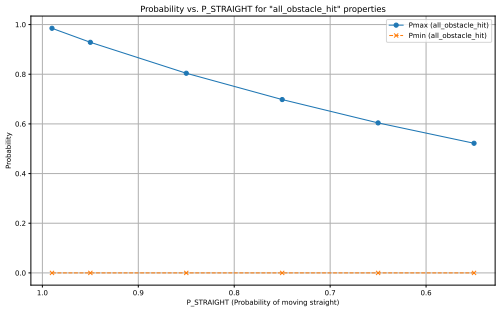



Figure 8: Probability of any drone hitting an obstacle.

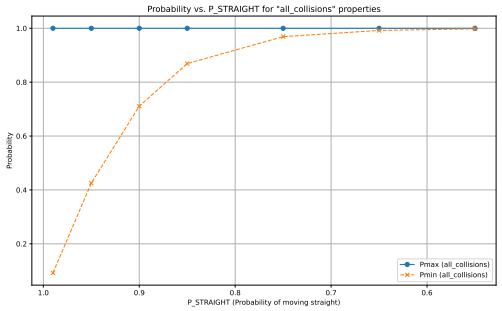



Figure 9: Probability of any inter-drone collision occurring.

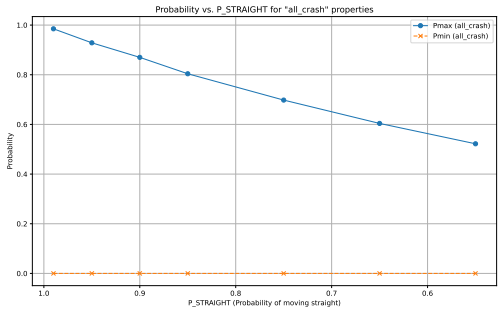



Figure 10: Probability of any drone crashing for any reason.

## 4.3 The Challenge of Scalability: State-Space Explosion

The most significant challenge encountered in the formal verification of multi-agent systems is the **state-space explosion** [6]. The size of the global state space, which the model checker must explore, grows exponentially with the number of agents and the complexity of their individual states.

This project provides a clear empirical demonstration of this phenomenon. Table 3 quantifies this growth. Increasing the number of drones from 3 to 4 on the same $5 \times 5$ grid causes the number of states to increase by a factor of approximately 270, from nearly 20 million to over 5.4 billion. This explosive growth makes exhaustive model checking computationally intractable for even modestly larger systems.
[1]

This scalability challenge also directly impacts mission performance. As shown in Figure 11, the maximum probability of all drones reaching their goals decreases as the number of agents increases. For a highly predictable system ($P_{\text{STRAIGHT}} \approx 0.99$), the success probability drops from 98% for a single drone to 95% for two drones, and down to 90.7% for three drones. The increased agent density makes finding safe, coordinated paths more complex, thus reducing the likelihood of total mission success.

[1] `https://github.com/dklgarima/Drone_swarm_coordination`

Table 3: State-Space and Transition Growth.

| Configuration | States | Transitions |
|---|---|---|
| 3 drones, $5 \times 5$ grid | 19,996,372 | 103,284,216 |
| 4 drones, $5 \times 5$ grid | 5,402,950,413 | 38,238,950,654 |
| 3 drones, $10 \times 10$ grid | 456,513,924 | 3,591,456,522 |

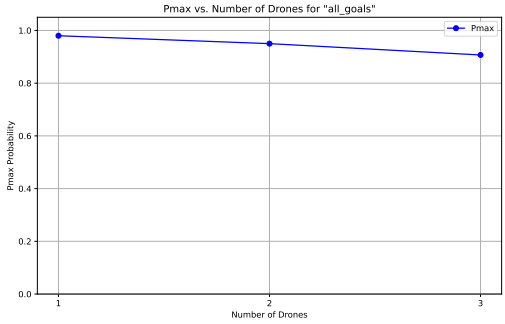



Figure 11: Maximum probability of all drones reaching their goals as a function of swarm size.

# 5 Conclusion and Future Work

This report successfully demonstrated the application of probabilistic model checking to the formal analysis of a decentralized drone swarm coordination protocol. By modeling the system as an MDP and using PRISM, we provided rigorous, quantitative guarantees about mission-critical properties. The analysis quantified the sharp decline in goal reachability as movement becomes less predictable and revealed a fundamental trade-off between optimality and robustness. Our results empirically confirmed that the state-space explosion is the primary barrier to scalability, with the analysis finding that a configuration of **3 drones on a** $5 \times 5$ **grid** was the most complex setup that remained tractable for verification.

The primary limitation identified is the state-space explosion. Future work should focus on addressing this challenge. This could involve exploring abstraction techniques to reduce model complexity or applying statistical model checking, which can provide high-confidence results without exhaustively exploring the entire state space. Further research could also extend the model to incorporate more realistic dynamics, such as time constraints or more sophisticated, adaptive agent behaviors, to bridge the gap between this formal model and real-world drone swarm applications.

# References


[1] Baier, C., & Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.

[2] Chrszon, P., Dubslaff, C., Klüppelholz, S., & Baier, C. (2018). ProFeat: Feature-oriented Engineering for Family-based Probabilistic Model Checking. *Formal Aspects of Computing, 30*(1), 45-75.

[3] Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A. (2011). Symbolic Model Checking of Software Product Lines. In *ICSE 2011*.

[4] Kwiatkowska, M., Norman, G., Parker, D. (2011). PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV 2011*.

[5] Soria, E., Schiano, F., Floreano, D. (2021). Distributed Predictive Drone Swarms in Cluttered Environments. *IEEE Robotics and Automation Letters, 6*(1), 104–111.

[6] Baier, C., Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.