Automated Discovery of Semantic Attacks in Multi-Robot Navigation Systems

Doguhan Yeke and Kartik A. Pant, *Purdue University;*
Muslum Ozgur Ozmen, *Arizona State University;*
Hyungsub Kim, *Indiana University Bloomington;*
James M. Goppert, Inseok Hwang, Antonio Bianchi,
and Z. Berkay Celik, *Purdue University*

https://www.usenix.org/conference/usenixsecurity25/presentation/yeke

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Automated Discovery of Semantic Attacks in Multi-Robot Navigation Systems

Doguhan Yeke[†], Kartik A. Pant[†], Muslum Ozgur Ozmen[‡], Hyungsub Kim[§]
James M. Goppert[†], Inseok Hwang[†], Antonio Bianchi[†], and Z. Berkay Celik[†]

[†] *Purdue University, {dyeke, kpant, jgoppert, ihwang, antoniob, zcelik}@purdue.edu*
[‡] *Arizona State University, moozmen@asu.edu* [§] *Indiana University Bloomington, hk145@iu.edu*

Abstract

Finding collision-free paths is crucial for autonomous multi-robots (AMRs) to complete assigned missions, ranging from search operations to military tasks. To achieve this, AMRs rely on collaborative collision avoidance algorithms. Unfortunately, the robustness of these algorithms against false data injection attacks (FDIAs) remains unexplored. In this paper, we introduce Raven, a tool to identify effective and stealthy semantic attacks (e.g., herding). Effective attacks minimize positional displacement and the number of false data injections by using temporal logic and stochastic optimization techniques. Stealthy attacks remain within sensor noise ranges and maintain spatiotemporal consistency. We evaluate Raven against two state-of-the-art collision avoidance algorithms, ORCA and GLAS. Our results show that a single false data injection impacts multi-robot systems by causing position deviation or even collisions. We evaluate Raven on three testbeds–a numerical simulator, a high-fidelity simulator, and Crazyflie drones. Our results reveal five design flaws in these algorithms and underscore the importance of developing robust defenses against FDIAs. Finally, we propose countermeasures to mitigate the attacks we have uncovered.

1 Introduction

Autonomous multi-robots (AMRs) consist of robots working cooperatively to complete tasks that would be difficult or impossible for a single robot. The use of multiple robots has become widespread in surveillance and industrial domains [5, 29, 46, 70]. While Amazon plans to conduct robot deliveries with the Amazon Prime Air service [6, 56], another notable real-world application is the tactical use of drone swarms in security operations [25]. Moreover, the concept of the *Internet of Drones (IoD)* [2] introduces a framework in which drones from different entities share and operate within the same environment in a coordinated and connected manner [30]. This paradigm enables easy integration of multiple drones, facilitating cooperative task execution [10].

A particularly concerning vulnerability of robotic systems is false data injection attacks (FDIAs), in which adversaries manipulate sensor readings or communication data to disrupt AMR operation. We focus on FDIAs because AMRs depend heavily on accurate communication data for safe navigation. When these data are compromised, attackers can cause significant disruptions that undermine both the safety and mission objectives. The attack vectors of FDIAs take many forms. For example, since March 16, 2024, drones in US airspace must comply with Remote ID regulations [61]. These regulations require drones to transmit their identity, position, and velocity. This information is crucial for safe navigation of both drones and other aircraft. However, Remote ID lacks both authentication and encryption, making it vulnerable to FDIAs. Similarly to Remote ID, adversaries can inject false data into multi-robot systems through ADS-B messages [17], physical sensor spoofing [33, 79], and insiders in IoD [8, 22].

The FDIAs mainly impact the behavior of the multi-robot collision avoidance (MRCA) algorithms of AMRs. These algorithms rely on sensor measurements for self-localization and broadcast messages to track surrounding robots [63, 72, 80]. Attackers can inject false data into these messages, disrupting the system's ability to ensure safe navigation. Such exploitation poses a critical risk to safety and functionality of multi-robot systems by causing potential physical damage (e.g., collisions with other robots or buildings).

Existing studies on robotic vehicle security can be broadly grouped into two categories: (*a*) single-robot systems and (*b*) multi-robot systems. A line of work [41–43] investigates single-robot systems. These studies focus on a single robot and do not address vulnerabilities in multi-robot navigation.

Another line of research [14, 73] explores autonomous vehicle (AV) and vehicle platooning [1, 18] vulnerabilities. Although related, the AVs domain differs from mobile robots due to different objectives, such as lane keeping, and different control systems with different degrees of freedom. Similarly, the vehicle platooning domain differs from mobile robots with its operation as a single unit along fixed lanes and in a rigid formation, and its leader-follower architecture.

Two recent studies have investigated the security of multi-robot systems, focusing on vulnerabilities in swarm-based navigation. SwarmFlawFinder (SFF) [34] uses an external drone to identify logic bugs in swarm control algorithms through fuzzing based on a counterfactual mutation strategy. SwarmFuzz (SF) [87] is a fuzzing technique that targets GPS spoofing attacks to cause collisions between a victim robot and a static obstacle.

While these works provide valuable insights and methodologies, our preliminary study (See Section 4) revealed limitations that make them less practical to extend for FDIAs against MRCA algorithms. To summarize, first, both works do not fully consider the complex dynamics of multi-robot interactions. That is, SFF's causality-based mutation strategy neglects the indirect interactions among multiple robots, while SF focuses solely on two-robot scenarios. This simplification overlooks the ripple effect phenomenon [90], which refers to how the actions of one robot can change the behavior of others in AMRs. A ripple effect is important as an attack on a single robot can trigger cascading failures and unexpected behaviors throughout the entire swarm (detailed in Section 4). Second, they consider limited attack spaces crucial for evaluating the MRCA algorithms. SFF's fuzzer relies on a restricted set of four actions (i.e., push back, follow, fragmentation, direction change), while SF constrains its analysis on only horizontal sensor spoofing. Consequently, they cannot be extended to adequately explore the full spectrum of potential attack parameters, including optimal sensor values, injection time, and duration. Adapting these studies to explore vulnerabilities in MRCA algorithms is challenging, as their attack strategies are tightly coupled with specific actions.

To address these gaps, we introduce Raven, a framework to find *semantic* attacks in multi-robots. We define a semantic attack as a form of adversarial manipulation that exploits the robot's interpretation of its environment, causing it to deviate from its intended task. To this end, we define five different semantic attack goals. We identify three of them from prior work (navigation delay, robot-robot collision, and robot-obstacle collision), and we newly discover two other goals (deadlock and herding). Achieving these goals is challenging due to high-dimensional input spaces and the need for the adversary to remain stealthy by evading the detectors deployed. For an attack to remain stealthy, the adversary must craft perturbations that are both minimal and contextually plausible. To address this, we first formally represent attacks using temporal logic formulas. We define spatio-temporal and stealthiness constraints that keep attack parameters within a distribution to evade existing detectors. To effectively find stealthy attack parameters, we employ a robustness-guided search method combined with a stochastic optimization technique.

We evaluate Raven with two state-of-the-art MRCA algorithms (ORCA [80] and GLAS [63]) on three testbeds: an interactive Python simulator, a PX4 and Gazebo-based simulator, and a swarm of Crazyflie drones. Our results show that even a single false data injection can significantly impact multi-robot systems, causing collisions and enabling shepherding (i.e., large lateral deviations) attacks. Raven achieves a high success rate of 90% to 100% with attacks that include only five injections. We conducted experiments with various attack parameters and realized them in realistic physics-based simulations and real-world experiments to demonstrate the feasibility of performing the FDIAs in practice. Finally, we analyze the vulnerabilities and categorize them into five different root causes (e.g., high reactivity and planning-time trade-off). In summary, we make the following contributions:

- We introduce two new adversarial objectives targeting multi-robot systems that disrupt their intended navigation: "deadlock", which immobilizes robots by maintaining their positions for a certain duration, and "herding", which directs robots into attacker-specified areas.
- We introduce Raven, a tool that leverages signal temporal logic to formally express our two new adversarial objectives and three existing goals from prior work, and systematically identifies optimal false data injection attack parameters that achieve these goals in MRCA algorithms.
- We evaluate Raven by targeting two state-of-the-art MRCA algorithms, ORCA and GLAS, within different testbeds in diverse environmental configurations, and assess its feasibility in real-world scenarios. Our analysis reveals five distinct design flaws that make them vulnerable to attacks.

We have responsibly disclosed our findings to the developers of ORCA and GLAS and shared our report with ten stakeholders, including multi-robot companies and agencies. Our project is available for public use and validation [88].

2 Background

Robot Software Stack. In this paper, we focus on aerial robotic vehicles, although we anticipate that the methodologies we introduce can be extended to ground and aquatic systems (detailed in Section 7). An aerial robotic software stack consists of four main components [72]: sensing, mapping and localization, planning, and control, as shown in Figure 1.

The *sensing* component enables the robot to sense its physical environment with RGB images from cameras and 3D point clouds from vision sensors (e.g., LiDARs and depth cameras). The *mapping & localization* component estimates the robot's physical state (e.g., position) using sensor measurements, including global navigation satellite systems (GNSS) and the inertial measurement units (IMUs). Raw sensor measurements from non-vision sensors are next passed to sensor filters such as low-pass [45] (e.g., configured with IMU_GYRO_CUTOFF in PX4 [58]) and dynamic harmonic notch filters [32]. Filtered

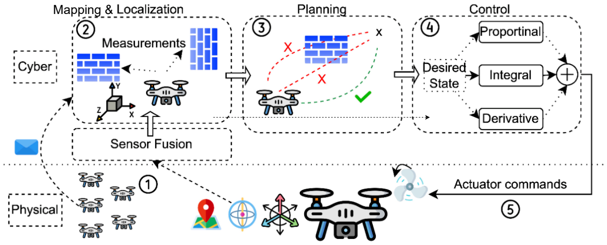

Figure 1: A robot software stack.

sensor readings are then passed to sensor fusion algorithms, such as extended Kalman filters [36] (EKFs), to estimate the physical states of a robot (e.g., position). The *planning* component then sets the robot's target physical states (e.g., the desired position). Lastly, the *control* component calculates the proper commands for the actuators (i.e., motors) to make the robot achieve the desired physical states.

Multi-Robot Coordination. Multi-robots achieve state awareness (position and velocity) of other robots through two primary methods: (*i*) vision sensor-based estimation and (*ii*) communication-based state sharing. In vision sensor-based estimation, robots rely on on-board sensors (e.g., LiDAR and camera) to perceive the states of other robots. However, this method faces several problems: (1) many vision sensor components employ machine learning algorithms (e.g., computer vision-based techniques) that lack formal proofs of correctness, and (2) vision sensors may return erroneous readings due to various reasons, from malfunctions to environmental conditions. These problems may cause sensor-based estimation to determine other robots' states incorrectly.

To address these concerns, communication-based state sharing that relies on data exchange between robots is broadly used. Each robot estimates its position and velocity through its sensors (such as GNSS and accelerometers) and shares its state with other robots using communication protocols. This method has several advantages: (1) it provides precise information about the location of all robots (up to the accuracy of the ego robot's state estimation), (2) robots do not need to depend on potentially error-prone sensing modules to estimate other robots' positions and velocities, and (3) this approach ensures that each robot maintains a consistent view of other robots' states in multi-robot systems.

Robots use different protocols to share their states. In particular, Micro Air Vehicle Link (MAVLink) [48], originally developed for open-source autopilots such as ArduPilot [7] and PX4 [58], is widely adopted by uncrewed aerial vehicles in academic and research environments due to its lightweight, packet-based messaging framework. Larger robots, such as aircraft, broadcast their states through Automatic Dependent Surveillance-Broadcast (ADS-B) [9].

Smaller uncrewed aircraft vehicles (UAVs), such as drones, use remote identification (Remote ID) [61]. Starting March 16, 2024, the Federal Aviation Administration (FAA) [27] requires that all drones broadcast their identity and position [62]. Although both ADS-B and Remote ID help maintain situational awareness, they are neither encrypted nor authenticated. As a result, robots send their state messages (i.e., `{robot_id, position, ...}`) in plaintext. These state messages are inputs to the collision avoidance algorithms.

Multi-Robot Collision Avoidance (MRCA). Collision-avoidance modules focus on immediate responses to dynamic environmental factors. These modules ensure the safety of robots by planning actions over a short time frame to avoid collisions. MRCA algorithms can be categorized into two types: centralized and decentralized [71].

Centralized MRCA algorithms rely on a ground control station (GCS) that collects information from all robots, such as their physical states (e.g., position), and computes trajectories for each robot. These algorithms require substantial computational power and are generally infeasible for resource-constrained systems, such as mobile robots with limited RAM and flash memory [72]. Therefore, these algorithms must be executed on GCS, with robots periodically updated about their next waypoint. As a result, centralized algorithms require constant communication between robots and the GCS, which is impractical over large areas or extended periods. In addition, this approach is vulnerable to a single point of failure.

In *decentralized* MRCA algorithms, each robot computes its trajectory using partial environment information. Thus, robots have limited knowledge of the environment and mainly rely on their sensors (e.g., GNSS) for their state estimation (e.g., position) and wireless communication protocols such as ZigBee, Bluetooth, and Wi-Fi [35] to obtain states from other robots in their vicinity. Decentralized algorithms guarantee fast computation and can be deployed onboard for real-time execution, making them suitable for resource-constrained systems due to size, weight, and power limitations.

Notable examples of collision-avoidance algorithms include optimal reciprocal collision avoidance (ORCA) [80], control barrier functions (CBFs) [82], pathfinding via reinforcement and imitation multi-agent learning (PRIMAL) [67], and global-to-local autonomy synthesis (GLAS) [63]. Developers and researchers commonly adopt and/or extend these algorithms for real-world applications such as car-like robots and ground mobile robots [4, 15]. We detail our selection criteria for ORCA and GLAS algorithms in Section 6.

3 Threat Model

We denote three robot roles: the *target robot*, whose position readings or messages are manipulated by the adversary; the *victim robot*, which the adversary seeks to attack; and *bystander robots*, which are all other robots in the system.

We consider two types of semantic attacks, *shepherding* and *collision* attacks. The shepherding attacks aim to (*i*) steer the victim towards an attacker-desired area or (*ii*) immobilize the robot for a certain duration, causing it to fail/delay its mission (e.g., time-sensitive deliveries). The goal of collision attacks is to cause collisions between robots or with obstacles while the target robot is not involved in the collision. Direct attacks (e.g., spoofing the victim robot's GNSS readings directly and making it collide with an obstacle) are outside the paper's scope, as prior studies examined such scenarios [23, 68].

Adversary's Capabilities. We consider an adversary who conducts FDIAs on fully autonomous multi-robots. The adversary executes FDIAs by manipulating the *target robot*, thereby aiming to affect the *victim robot*.

Figure 2 provides three concrete scenarios that depict three representative FDIAs: (1) insider/intruder (left), (2) Remote ID or ADS-B spoofing attacks (center), and (3) physical sensor attacks such as GNSS or IMU spoofing (right).

First, insider attacks can occur due to malicious robot(s) within the system, such as in *Internet of Drones (IoD)* [2, 30], which are gaining popularity as they allow a vast number of UAVs to share the same airspace and to collaborate on complex missions such as search and rescue operations [59, 64]. However, these heterogeneous systems allow adversaries to introduce a malicious robot that can transmit fake position and velocity to other robots in the swarm [8, 22]. Figure 2 (Left) demonstrates the insider/intruder attack vector, where an adversary controls a robot (acting as the target) to directly inject false position ($\{p_a\}$) into the multi-robot communication network, thereby manipulating the information used by the victim robot for collision avoidance.

Second, UAVs broadcast Remote ID [61] or ADS-B [9] messages, which is mandated by FAA [27]. Yet, these messages are neither authenticated nor encrypted [17, 74]. Thus, the attacker can spoof these messages that include their identification, position, and velocity while operating in the airspace. Figure 2 (Middle) shows Remote ID or ADS-B spoofing, where the attacker broadcasts spoofed messages ($\{p_a\}$) containing fabricated position data of the target robot, which are then received and processed by the victim robot.

Third, an attacker can conduct a physical sensor attack (e.g., spoofing GNSS signals) to make the target robot obtain the spoofed location and report the false position to other robots, including the victim robots [23, 68, 79]. Furthermore, targeted GNSS spoofing, which uses directional antennas to target a single GNSS receiver among multiple receivers, has shown a significant impact both theoretically [79] and experimentally [33] in multi-robots. Figure 2 (Right) shows that the adversary conducts sensor spoofing by generating fake GNSS signals that deceive the target robot's sensors, causing the target robot to calculate and report an incorrect position ($\{p_a\}$) to all robots, including the victim robot.

Adversary's Knowledge. We assume that the adversary knows MRCA algorithm running on robots, as well as their initial and target positions. We assume that robots do not have visual sensors (e.g., cameras) to cross-validate the integrity of incoming messages, similar to previous work [86, 87]. This assumption aligns with real-world deployments—our analysis of 32,190 PX4 flight logs shows only 0.8% of UAVs use vision sensors, likely due to battery constraints [12].

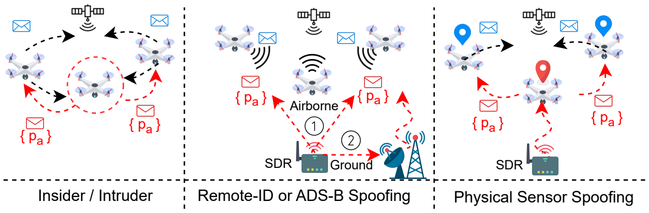

Figure 2: Attack vectors of FDIAs. (Left) Insiders or intruders directly inject false data into the network. (Middle) Attackers spoof Remote ID or ADS-B signals through (1) targeting line-of-sight drones equipped with receivers or (2) spoofing ground stations that relay messages to aircraft. (Right) Attackers manipulate physical sensors to generate false readings propagating through the network.

4 Motivation and Challenges

4.1 Motivating Example

We show an example of a shepherding attack on AMRs that causes navigation delay in a search and rescue operation.

Benign Case. Figure 3 (Left) demonstrates the benign case in which three robots navigate to their respective goal positions while avoiding collisions. The robots first pass near a block of houses (obstacle 1). They next have to pass a tree (obstacle 2) by avoiding any collisions. Since robots are fully autonomous, they dynamically decide their target waypoints (i.e., each dot) at each time step rather than using static waypoints. To decide these dynamic waypoints, each robot follows three steps: (1) Each robot calculates its estimated position and velocity through its sensor measurements (e.g., GNSS, accelerometer). (2) Each robot sends its position and velocity to all other robots. (3) The algorithm deployed decides the robot's velocity and the target waypoint for the next time step.

Navigation Delay Attack. Figure 3 (Middle) demonstrates the adversarial case in which the attacker aims to cause a navigation delay for the victim robot (shown as ⓪). To achieve this, the attacker performs a Remote ID spoofing attack at time step t on the target robot (shown as ①). When the attacker spoofs the target robot's Remote ID broadcast at time t (i.e., t_a), the victim robot receives inaccurate position data about the target robot (x_a=x_3, y_a=y_2). Based on the spoofed position broadcast, the victim robot (⓪) calculates its target waypoint and velocity for the next time step at time t+1 (Right). The spoofed Remote ID broadcast creates a deviation of $x = -1.2$ and $y = 1.9$ in the victim robot's route, steering it to the left side of obstacle 1 (red dotted line). The victim robot has two

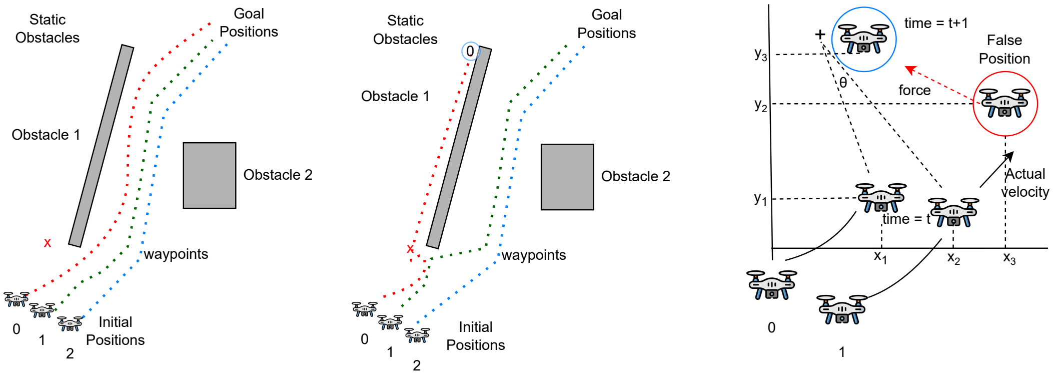

Figure 3: Illustration of a navigation delay attack in an autonomous multi-robot Search and Rescue (SAR) scenario. (Left) Under benign conditions, both robots (R0, the "victim", and R1, the "target") follow optimal trajectories toward their goal positions. (Middle) The compromised (target) robot R1 maneuvers so that R0 is forced into the left side of an obstacle (shown as a red X). (Right) Specifically, this is achieved by causing R1's position in front of R0 (shown by waypoints x3, y2), causing R0 to deviate from its original path. As a result, R0 detours around the obstacle via a longer, suboptimal route in timely SAR operations.

options: wait for the target robot to clear its optimal path or take a longer alternative route. Unlike autonomous vehicles, these robots rely on local planners and occasional support from a ground control station. Consequently, the victim robot takes the longer path, more than doubling its navigation time.

Figure 3 (Right) demonstrates that the adversary must plan three important attack parameters: when to attack ($\mathbf{t_a}$), how to attack ($\mathbf{x_a}$, $\mathbf{y_a}$), and how many times to inject false data ($\mathbf{n_a}$). For t_a, the adversary specifically conducts the attack at the strategic time when the robots are passing close to obstacle 1. For x_a and y_a, the adversary calculates the displacement required to steer the victim drone to the left side of the obstacle. To remain stealthy (detailed in Section 7), the position errors should be within a specified range (e.g., 5 m [31]) to avoid being filtered out by the target robot's deployed detector. For n_a, the adversary spoofs the target robot once, avoiding unnecessary spoofing (detailed in Section 7). It is challenging to systematically find these optimized attack parameters. Unfortunately, the prior work on multi-robot attacks [34, 87] cannot identify these attack parameters. This is because they limit their search space to a limited set of parameters, whereas identifying this attack's parameters requires searching for continuous values, which they do not support.

4.2 Design Challenges

C1: High-Dimensional Attack Parameter Search. Finding optimal attack parameters requires exploring a vast multidimensional space. This space includes when to launch the attack, how long to maintain it, and how to manipulate both the *x* and *y* displacements when crafting false data. Prior work restricts the search to four predefined maneuvers or to horizontal spoofing, which limits the search space and reduces the attack success rate [34, 87]. Prior research also focuses on a single target-victim robot pair, which overlooks indirect ripple effects [87], which refers to how the actions of one robot can alter the behavior of others in a multi-robot system. For example, as shown in Figure 3, an injection attack on Robot 2 can affect Robot 1, which in turn impacts Robot 0. Consequently, small position changes in one robot can alter the next waypoint calculations of other robots. This complexity grows with the number of robots, intensifying the challenge of finding accurate attack parameters.

C2: Attack Generalization. Prior work focuses on a single scenario with only one attacker's goal (i.e., a robot-obstacle collision) in an environment [87]. However, integrating their approaches into other attacker goals is not feasible for two main reasons. First, their optimization function is only tailored to a distance metric that cannot be easily extended to shepherding attacks. Second, their gradient-guided optimization assumes a convex objective function that minimizes the distance between a victim robot and an obstacle. However, this assumption fails, as attacks in multi-robot systems often include non-convex objectives as detailed in Section 5. In addition, their approaches are not extendable to multiple victims, multiple attackers, or environments with multiple obstacles, due to their linear optimization function to calculate the distance between a robot and an obstacle. However, this severely constrains the search space for the attacker.

C3: Attack Detection. Previous work applies constant spoofing with a fixed displacement or requires physical proximity to an external drone, both of which can be easily flagged by deployed detectors [34, 87]. Instead of constant spoofing, perturbations should be crafted to be small to minimize the likelihood of detection. Thus, the attacker must find minimal perturbations to achieve the attack goal. These injections must also remain spatio-temporally consistent. For example, the attacker should remain within the velocity constraints of MRCA algorithm (e.g., ORCA [80]) to avoid apparent anomalies.

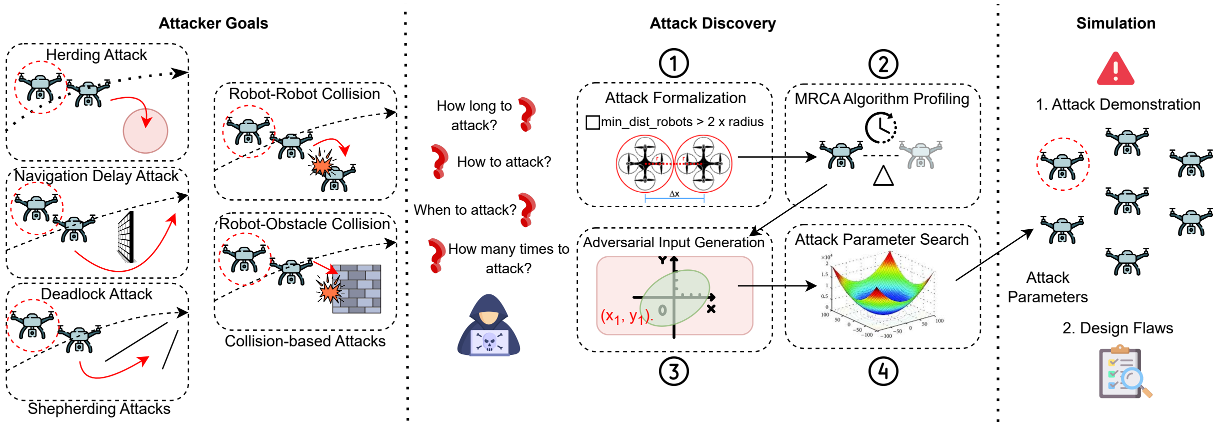

Figure 4: Overview of Raven's architecture.

C4: Real-World Impact. Prior studies often rely on MATLAB/R-based simulators [34, 76, 87]. However, these simulators lack full integration of autopilot systems (e.g., PX4 [58]), MRCA algorithms, and real-time multi-robot communication. This impacts the effectiveness of attacks in practice. To demonstrate practicality, attacks should be evaluated on Software-in-the-Loop (SITL) simulation with Autopilot programs (e.g., PX4 [58] and Ardupilot [7]). However, current simulators do not fully support collective multi-robot navigation. This is challenging as it requires distributed coordination (i.e., separate Autopilot deployment for each robot) and real-time multi-robot communication.

5 RAVEN Design

We present Raven, a tool that integrates a search-based method with a stochastic optimization function to identify feasible attacks on MRCA algorithms.

Approach Overview. Figure 4 illustrates the architecture of Raven, which operates in four phases: (1) defining attacker goals and formally specifying them for vulnerability analysis of the MRCA algorithms, (2) profiling the MRCA algorithms for attack synthesis, (3) generating adversarial inputs under physical constraints, and (4) conducting a robustness-guided search to determine feasible attack parameters.

We first formally represent the attacker's goals using Signal Temporal Logic (STL) formulas (①). We next integrate the attack parameters directly into the predicates of the STL formulas, making Raven adaptable to varying conditions, such as selecting different victim and/or target robots (**C1**, **C2**).

We conduct an initial run without any attack to observe the robots' trajectories in a benign scenario (②). This step offers two main insights to Raven. First, it allows us to create a pool of potential target-victim robots. Particularly, we record the timestamps when the robots pass near obstacles. This information helps narrow the search space for the time parameter, making the attack design more efficient (**C1**). Second, it provides the total simulation time of the mission, which is used to sample the time parameter within this range.

We enhance attack stealthiness by addressing spatio-temporal constraints and adhering to GNSS tolerance thresholds to evade commonly used detectors (③). We generate false messages within physical constraints to reflect realistic position changes and remain within GNSS noise thresholds (**C3**). Additionally, we implement an intermittent attack strategy by sampling attack times from the total simulation duration to reduce the likelihood of detection by commonly deployed detectors (detailed in Section 5.3).

We model each MRCA algorithm as a black-box component and incorporate it into Raven (④). A cost function is defined for each attack type, and the stochastic optimization function iteratively generates attack parameters to identify those required for successful false data injections (**C2**). The stochastic optimization function is critical for STL, as STL has non-linear and non-convex properties.

5.1 Attack Goal Formalization

Table 1 outlines the formalization of each attack. The column "Attack Type" lists the attack's name. The column "Explanation" provides a description of the attack. The column "STL Formula" formally represents each attack using a Signal Temporal Logic (STL) formula. We choose STL over other temporal logics for two reasons. First, multi-robot systems have continuous state variables, such as positions and velocities. Second, STL allows for precise timing constraints, such as "the robot should stay in this region for 10 seconds."

Herding. We define `dist of victim to point` as the distance of the victim robot to a specific location chosen by the attacker. The formula states that the distance from the victim to the attacker-desired location must always be greater than x.

Deadlock. We define `total pos diff of victim` as the position displacement of the robot given a certain time period. The formula states that within the mission time, the total

Table 1: Description of STL specifications for attacks on MRCA algorithms.

ID	Attack Type	Explanation	STL Formula
Shepherding Attacks			
A1	Herding	The target robot causes the victim robot to navigate into the attacker-desired area (gets closer by x).	$\square(\text{dist\_of\_victim\_to\_point} > x)$
A2	Deadlock	The target robot causes the victim robot to be stuck in a position (no position change more than a threshold (x) for a certain time (t)).	$\square_{[t,\text{mis\_time}]}\ (\text{total\_pos\_diff\_of\_victim} > x)$
A3	Navigation Delay	The target robot causes the victim robot to navigate an extended distance and increase navigation time (d times) without causing deadlock.	$\Diamond_{[0,\text{mis\_time}\times d]}\ (\text{victim\_pos} = \text{goal\_pos}) \wedge \square_{[t,\text{mis\_time}]}\ (\text{total\_pos\_diff\_of\_victim} > x)$
Collision Attacks			
A4	Robot-Robot Collision	The target robot causes the victim robot to collide with another robot in the multi-robot system.	$\square(\text{min\_dist\_victim\_robot} > 2 \times \text{robot\_radius})$
A5	Robot-Obstacle Collision	The target robot causes the victim robot to collide with a static obstacle in the environment.	$\square(\text{min\_dist\_victim\_obst} > \text{robot\_radius})$

† □ refers to “always”, ◊ refers to “eventually”.

position change of the victim robot in every time window t must always be greater than x. We illustrate the deadlock attack in Figure 12 in Appendix A.

Navigation Delay. The formula means that the victim robot "eventually" reaches the goal position over an extended time period. Here, we define `d` based on the prior work [34]. The formula states that within the extended mission time, the total position change of the victim robot must always be greater than x, and the victim robot must always be in the target position at the end of the mission.

Robot-Robot Collisions. We define `min dist victim robot` as the minimum distance between the victim robot and other robots, and `radius` is the radius of the robot. The formula states that the minimum distance between the victim robot and another robot must always be greater than twice the `radius`. In our framework, we exclude the target robot from direct collision scenarios with the victim robot for subsequent attacks. We motivate this approach with two main reasons: (1) minimizing liability and (2) preserving the target robot for future attacks during the mission, as detailed in Section 7.

Robot-Obstacle Collisions. We define `min dist victim obst` as the minimum distance between the victim robot and its closest obstacle. The formula states that this distance must always be greater than the robot's radius.

5.2 MRCA Algorithm Profiling

Algorithm 1 presents the `Raven`'s steps, with each component represented by a corresponding function. `Raven` begins with MRCA algorithm profiling.

To craft attack messages, `Raven` requires attack ranges for each parameter, as the search space is significantly large. To determine these ranges, we run the MRCA algorithm without injecting any attacks to observe the total navigation time. We then sample the time parameter from this range $[0,\ t_{\text{sim}}]$.

Here, we model MRCA as a system under test (SuT) that outputs time-series data (Line 4). Specifically, the model takes start and goal positions of the robots as input, and outputs system trajectories as follows:

$$M : (S, G \to \mathbb{R}^n) \to (W \to \mathbb{R}^s) \tag{1}$$

where S and G represent start and goal positions, W denotes robot waypoints, n is the number of robots, and s is the simulation duration divided by timestep Δt.

Next, we identify the target and victim robots. We calculate robot-obstacle distances for each robot-obstacle pair (e.g., $[O_1\text{: } [R_2\text{:1.2m, } R_3\text{:1.4m,}]]$). We then populate it with potential target-victim pairs. Specifically, robots passing closest to obstacles are prioritized as victims, and the robot closest to victim is selected as target. This approach reduces the time complexity of searching for victim-target pairs in the system.

In addition, we record timestamps when robots pass near obstacles. This information can further narrow the search space for the time parameter for attacks (e.g., collision), making the attack design more efficient. It is important to note that no attack is conducted during this run; the attacker's goal is to collect data to craft effective attacks in this environment.

5.3 Adversarial Input Generation

When generating parameters, we set two constraints to enhance attack feasibility and reduce the likelihood of detection.

Spatio-temporal Consistency. The spatio-temporal consistency enables the attacks to be physically viable and plausible. For example, if an attacker creates a sudden jump in the GNSS coordinates of the system, it can be trivially flagged by a runtime monitor, which checks the consistency of the GNSS measurements over time. In this work, the attack parameters are designed based on the feasibility of the false message. During the crafting of the attack message, MRCA preserves spatiotemporal consistency within the physical constraints. Specifically, the position difference of the robots between consecutive time steps should obey physical constraints. Given the position difference, the robot can reach the next target waypoint within the speed limits of the robot:

$$p^j(k+1) = p^j(k) + \Delta p^j(k) \tag{2}$$

Algorithm 1 Attack Synthesis

```
 1: Input: MRCA Algorithm Profile ($\mathtt{A_p}$), STL Spec. of Attack Goal($\varphi_a$)
 2: Output: Attack Parameters ([($\mathtt{t_a}$, Δx, Δy), ...]) || Timeout ($\mathtt{t_{out}}$)
 3: procedure ATTACKSYNTHESIS
 4:     $\mathtt{R_{ins}}$ = initMRCA(), $\mathtt{A_{ins}}$ = initAttacker()
 5:     $\mathtt{A_{ins}}$ → [applySpatioTemporal(), applyStealthiness()]
 6:     for $\mathtt{n_{inj}} \in [1, \mathtt{max_{inj}}]$: do
 7:         for $\mathtt{it} \in [1, \mathtt{num_{it}}]$: do
 8:             ($\mathtt{t_a}$, Δx, Δy) = $\mathtt{A_{ins}}$ → planAttack($\mathtt{n_{inj}}$)
 9:             $\mathtt{R_{ins}}$ → applyAttackParams($\mathtt{att_{id}}$, $\mathtt{t}$, $\mathtt{P_{all}}$, $\mathtt{V_{all}}$)
10:             $\mathtt{T_{all}}$ = $\mathtt{R_{ins}}$ → calculateNextPositions()
11:             $\mathtt{score_r}$ = $\mathtt{A_{ins}}$ → evaluateRobustness(traces)
12:             $\mathtt{A_{ins}}$ → updateAttackParams($\mathtt{score_r}$)
13:             if $\mathtt{score_r} < \tau_r$: then
14:                 $\mathtt{att_{found}}$ = true
15:                 $\mathtt{A_{ins}}$ → evaluatePerformanceMetrics()
16:                 $\mathtt{A_{ins}}$ → visualizeRobotScenarios()
17:                 return [($\mathtt{t_a}$, Δx, Δy), ...]
18:             end if
19:         end for
20:     end for
21: end procedure
```

where $p^j(k)$ is the position of robot j at time k, and $\Delta p^j(k)$ is the position displacement (i.e., upper-bound attack effectiveness) by spoofing calculated by:

$$\Delta p^j(k) \leq t_s * v_{max} \tag{3}$$

where t_s is the sampling time and v_{max} is the maximum velocity set by the MRCA algorithm. Therefore, the attacker's maximum change in position stays within the physical constraints defined by the running algorithm. We apply these constraints in our search function in Section 5.4 (Line 5).

Evading Attack Detection. Another essential property of our attack design is evading attack detection. To tackle **C3**, we observe that most attack detection algorithms rely on thresholds to distinguish anomalies from normal signals. These thresholds are chosen to balance false alarms and detection accuracy. Therefore, these algorithms provide opportunities for attackers to inject false data into the system (i.e., hiding the attack below the noise floor and mimicking it as a disturbance). By altering the measurements, we can cause deviations over time to the estimated state of the system. In this work, we consider point-wise Chi-squared and CUSUM-based detectors [51]. (1) Chi-squared detection utilizes measurement residuals, the difference between predicted and measured outputs. For the i^{th} sensor at time step k, the residual $r_i(k)$ is:

$$r_i(k) = y_i(k) - \hat{y}_i(k) \tag{4}$$

Under nominal conditions, these residuals follow Gaussian distributions, with their energy following Chi-squared distributions. (2) CUSUM-based utilizes a recursive accumulation of residuals:

$$S_{i,k} = \begin{cases} \max\big(0,\, S_{i,k-1} + r_{i,k} - b\big) & \text{if } S_{i,k-1} \leq \tau, \\ 0 \quad \text{and} \quad \tilde{k} = k-1 & \text{if } S_{i,k-1} > \tau. \end{cases} \tag{5}$$

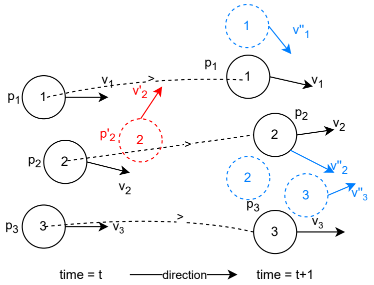

Figure 5: Dynamic waypoint calculation under FDIAs.

where $S_{i,0} = 0$, b is a tunable constant to keep the sum bounded, and τ is the threshold. An attack is considered stealthy if the test statistics remain within the threshold even in the presence of false data in the measurements. We integrate these constraints into the search process in Section 5.4. We evaluate different threshold values in Section 7.

5.4 Robustness-Guided Attack Search

We search for parameters that falsify the STL formulas (defined in Section 5.1). After defining the constraints, we need to search for optimal attack parameters. By "optimal", we mean minimizing the number of injections while ensuring the constraints hold. To address the challenges (**C2**, **C3**), we leverage a search-based method. We use the MRCA algorithm as a model to simulate the attack parameters, which are [$\mathrm{t_a}$, ..], [($\bar{x}$, $\bar{y}$), ..]. We define $\mathrm{t_a}$ as the time to attack, and ($\bar{x}$, $\bar{y}$) as the position placement at the attack time as follows:

$$x_{t_\mathrm{a}} = x_{t_\mathrm{a}} + \bar{x}_{t_\mathrm{a}}, \qquad y_{t_\mathrm{a}} \;= y_{t_\mathrm{a}} + \bar{y}_{t_\mathrm{a}} \tag{6}$$

Similarly to the profiling of the MRCA algorithm (now injecting false data), the model takes these states (i.e., x_{t_a}, y_{t_a}, t_a) as inputs and outputs system trajectories as follows:

$$M : (S \to \mathbb{R}^n) \to ([T, I] \to \mathbb{R}^s) \tag{7}$$

where S is the state to be tested and n is the number of FDIs. T are the system trajectories as a time series, I is time interval, and s is total simulation time divided by time step (Δt).

`Raven` begins searching with random values for time and position displacement parameters for each attack type, aiming to violate the requirements represented as temporal logic (Line 8). At each step, the model returns the time-series trace of the simulation. The framework analyzes these traces and generates new attack parameters.

Figure 5 illustrates that the attacker targets Robot 2 by injecting spoofed data, shown as a red-dashed circle. As a result, the robots' next positions at `t+1` (solid circles) deviate to new locations, represented by blue-dashed circles. This scenario repeats whenever `Raven` generates and injects false data for each iteration (Line 9). After each iteration, we use a stochastic optimizer to find the attack parameters, which aim to minimize the cost function (Line 12). The cost functions are defined based on the STL formulas (φ) shown in Table 1, e.g., the cost function of an A5 attack is:

$$\text{cost} = \|\mathbf{r}_{\text{center}} - \mathbf{o}_{\text{center}}\| - \left(r_{\text{agent}} + r_{\text{obstacle}}\right). \tag{8}$$

In this expression, $\mathbf{r}_{\text{center}}$ and $\mathbf{o}_{\text{center}}$ are the 2D positions of the agent and obstacle centers, respectively, and $\|\mathbf{r}_{\text{center}} - \mathbf{o}_{\text{center}}\|$ represents their Euclidean distance. The terms r_{agent} and r_{obstacle} are the bounding radii of the agent and obstacle, respectively. Subtracting these radii from the center-to-center distance computes the minimal distance between the agent and the obstacle surfaces: a positive value implies separation, zero indicates contact, and a negative value indicates overlap. This way, the attacker measures the robustness of false data injections against an STL formula quantitatively (Line 13). Once Raven identifies input parameters that cause a violation, the iterations end and return the parameters.

6 Implementation

To simulate our attacks in a realistic environment, we reviewed the existing literature and identified several limitations in current simulators. Initially, we examined flight controller simulators (Ardupilot [7], PX4 [58], and Paparazzi [55]), but found that they lack comprehensive support for multi-robot testbeds. We also evaluated the open-source tool, SwarmLab [76]. However, it only offers a MATLAB [47] simulator for swarms, which lacks high fidelity. Instead, Gazebo and PX4 Autopilot integration provides high-fidelity physics simulation, including accurate modeling of sensor noise, dynamics, and Autopilot programs. Furthermore, SwarmLab integrates only the Olfati-Saber [53] and Vicsek [81] algorithms, which are formation algorithms not specifically designed for multi-robot collision avoidance. Given these limitations, we developed a framework by extending an existing simulator [54] designed for a single UAV. To extend the existing simulator, we write 2850 LoC in Python and C++ (detailed in Appendix B).

We use the latest version of PX4 Autopilot firmware. We deploy a separate instance of PX4 on each robot, aligned with real-world use cases. To use PX4 in our framework, we write 1360 LoC in Python and C++.

We use PSY-TaLiRo [78], a Python package designed for the search-based test generation of Cyber-Physical Systems (CPS). PSY-TaLiRo offers a modular toolbox that supports multiple temporal logic monitors and optimization engines.

We use two state-of-the-art MRCA algorithms, ORCA [80] and GLAS [63]. These algorithms demonstrated how they outperform current approaches in terms of robot success rate and control effort [63]. We wrap these algorithms in Python to use them as a model in our Raven framework. We detail how ORCA and GLAS are selected and how Raven can be extended to other MRCA algorithms in Appendix A.

7 Evaluation

We evaluate Raven on three testbeds: the Python simulator, PX4/Gazebo-based high fidelity simulator, and Bitcraze Crazyflie 2.0 quadrotors. We utilize four distinct maps that represent real-world scenarios of multi-robots, similar to the evaluation setups of previous studies [34, 63, 80]. We evaluate Raven to answer the following research questions:

RQ1 What is the performance of Raven in finding successful attacks against MRCA algorithms?

RQ2 What is the performance of Raven under different configuration parameters (e.g., GNSS noise)?

RQ3 Can Raven evade the commonly deployed detectors in the flight controller? (stealthiness)

RQ4 What are the root causes of the attacks?

RQ5 What is the performance of Raven compared to baseline approaches?

RQ6 What is the execution time of Raven?

We run Raven on a computer with an i7-6850k 3.60GHz CPU, 48GB RAM, running 64-bit Linux Ubuntu 22.04.

Evaluation Metrics. We evaluated the attack success of Raven using four metrics. (1) For navigation attacks, we employed a *timeliness* metric that measures the robot's ability to reach its target position. An attack is considered successful if the robot fails to reach the goal within twice the benign completion time, following prior work [34]. (2) For herding attacks, we determine the success by *spatial deviation*, specifically when the victim robot enters an attacker-designated region that remains unvisited under a benign operation. (3) The effectiveness of the deadlock attack is measured by *mobility* analysis, where success is achieved if the victim robot becomes immobilized (position change < 0.1m) for ten consecutive time steps (i.e., 5 seconds). (4) For collision-based attacks, we use a *proximity* threshold where success is defined by the minimum distance between entities (robot-robot or robot-obstacle) reaching 0, which indicates physical contact.

7.1 Effectiveness (RQ1)

First, we evaluate Raven's performance in finding attacks against ORCA and GLAS. Next, we present two case studies, a herding attack and an end-to-end real-world attack demonstration with Crazyflie drones. Lastly, we provide a case study that illustrates a multiple-victim robot attack in Appendix A.

Table 2 shows the effectiveness of Raven in identifying attacks. The column "Attack Goal" specifies five distinct attack goals. "Benign Case" represents scenarios without any attack. For benign and adversarial cases, we performed 10 runs and documented the results. The column "Attack Discovery" indicates whether Raven successfully identifies the attack parameters, such as attack time and position displacement. In these experiments, we conducted 5 injections.

Raven consistently identified all attack parameters in both algorithms. In only one instance, the "herding attack" on

Table 2: Performance of Raven in finding attacks against two representative MRCA algorithms.

Attack Goal	Benign Case	Attack Discovery	Min # Injections	Attack Plan Time	Root Cause[†]
		Experiments on ORCA			
Robot-Robot Collision	0/10 (0%)	10/10 (100%)	1	2.38 s / 2.6 s / 2.94 s	HR-ICM-PTT-FC
Robot-Obstacle Collision	0/10 (0%)	10/10 (100%)	1	2.5 s / 4.2 s / 4.6 s	HR-ICM-PTT-FC
Herding	0/10 (0%)	10/10 (100%)	1	1.97 s / 2.26 s / 2.53 s	HR-ICM-PTT
Deadlock	0/10 (0%)	10/10 (100%)	1	1.22 s / 2.2 s / 2.44 s	HR-ICM-PTT
Navigation Delay	0/10 (0%)	10/10 (100%)	1	1.01 s / 3.35 s / 5.63 s	HR-ICM-PTT
		Experiments on GLAS			
Robot-Robot Collision	0/10 (0%)	10/10 (100%)	1	7:39 s / 7:58 s / 8:54 s	ICM-PTT-LA
Robot-Obstacle Collision	0/10 (0%)	10/10 (100%)	1	8:4 s / 10:2 s / 14:8 s	ICM-PTT-LA
Herding	0/10 (0%)	9/10 (90%)	3	2:35 s / 2:4 s / 2:42 s	ICM-PTT-LA
Deadlock	0/10 (0%)	10/10 (100%)	3	1:54 s / 2:44 s / 2:52 s	ICM-PTT-LA
Navigation Delay	0/10 (0%)	10/10 (100%)	3	2:15 / 2:22 s / 2:36 s	ICM-PTT-LA

[†] HR: High Reactivity, ICM: Imperfect Communication and Measurements, PTT: Planning vs. Time Tradeoff, LA: Learning-based Algorithms, FC: Feasibility of Collisions.

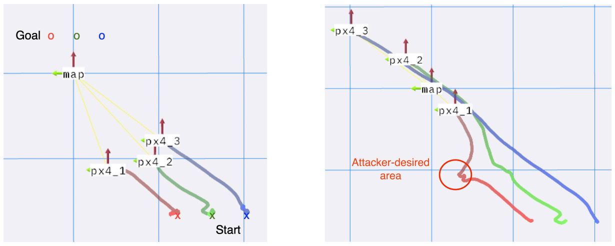

Figure 6: Demonstration of the herding attack on ORCA: (a) the benign scenario, where the robots navigate toward their goal positions, and (b) the attack scenario, where the victim robot is guided into an attacker-defined zone.

GLAS caused the tool to time out. However, increasing the number of runs eventually led to successful attack identification. Additionally, we iteratively reduced the data injection parameters when Raven detected an attack, aiming to identify the minimum number of false messages required to achieve the attacker's objective, as shown in Algorithm 1. The column "Min # Injections" represents the minimum number of false data injections needed for a successful attack.

Case Study 1 (Fig. 6) - *Herding Attack.* Raven discovers a herding attack on ORCA in the PX4/Gazebo simulator, as shown in Figure 6. Initially, three robots navigate to their respective goal positions (top-left) from their starting positions (bottom-right) while avoiding inter-robot collisions (Left). At t=12.1, the false data injection causes px4_1 robot to recalculate its target velocity to avoid a collision (Right). As a result, px4_1 robot veers off toward the attacker-designated area (shown as a circle). After the attack, the victim robot attempts to return to its optimal path. This shows how a victim robot can be shepherded into a specific zone.

Case Study 2 - *Experiment with Crazyflie* 2.0 *Quadrotors.* We conducted a real-world experiment using Crazyflie 2.0 quadrotors to demonstrate an end-to-end attack. Our setup includes three main components: (1) a motion capture module to receive real-time marker positions of Crazyflies, (2) the MRCA algorithm to calculate dynamic waypoints, and (3) an off-board module to transmit velocity commands to Crazyflies. We used a 6m × 6m indoor environment equipped with six Qualysis Motion Capture Cameras. We deployed three Crazyflies. We set the MRCA's time step to 0.5. Figure 11 in Appendix A demonstrates the test environment. In the benign run, robots navigated around the obstacle on the right side. Under the shepherding attack, the target robot induced the victim robot to deviate from its intended path and guided it onto the longer route. A demonstration video is available on the project website.

7.2 Analysis of Different Parameters (RQ2)

We tested different parameters, varying from operator errors in false data injection to different GNSS tolerance thresholds in our Python simulator. We present further auxiliary experiments in Appendix A.

Attack Reproducibility with Spoofing Time Delays. To test reproducibility with delays in injection, we first identified an attack using a single FDI with Raven. The attacker executed a "deadlock attack" with parameters: time = 6.0 s, x = -0.26, y = 2.20. We then examined how spoofing time delays affected the attack's success by sampling delays from $[-2,+2]$ seconds and running the experiment five times. The attack was successful 3 out of 5 times. Successful attacks occurred at t = 4.0 and t = 5.0, while failed attempts at t = 7.0 and t = 8.0 showed the robots had passed the obstacle and could not be placed in a deadlock position.

Different Target and Victim Robots. We demonstrated that Raven effectively finds the attack parameters for various target and victim robots. In Env3, we chose Robot 2 as the victim instead of Robot 0, which was the victim in Env2. Raven successfully identified the optimal attack parameters under different environmental conditions and with different victim robots. In addition, we tested scenarios with multiple target robots while finding attack parameters. We observed that having multiple target robots required fewer iterations and fewer false data injections. Thus, having multiple target robots can facilitate attack identification in terms of the time required to find the attacks and the number of injections needed.

We evaluate every target–victim combination, labeling each as feasible (optimal or suboptimal) or infeasible. For each pair, we execute a shepherding attack and record the number of FDIs required. Table 5 in Appendix A summarizes these results. From this experiment, a key result we found is that attacks require substantially fewer FDIs (i.e., 5 messages) when the target and victim robots are adjacent, while the presence of intermediate robots between the target and victim increases the number of FDIs (i.e., 10 messages). Raven iterates over all pairs and returns the one minimizing FDIs for each attack.

Attacks with Varying Number of Robots. We evaluated the scalability of attacks on varying numbers of robots. We extended our evaluation to scenarios involving n={2,4,8} robots in two different environments. In each scenario, the adversary controls one target robot to affect one victim robot. Raven successfully identified effective attack parameters for these *n*-robot configurations, including those with smaller (n=2) and larger (n={4,8}) numbers of robots.

We formalize the compromise ratio $\alpha = |\mathcal{T}|/|\mathcal{R}|$, where $\mathcal{T}$ denotes the number of targets that the adversary controls and $\mathcal{R}$ the set of all robots. We demonstrate attacks in scenarios where the compromise rate varies from 50% to 12.5%. Raven's attacks remain effective even at low compromise ratios, finding both collision and shepherding attacks. Our results show that successful attacks can often be launched by an adversary controlling only a single target $\mathcal{T} = 1$. Demonstration videos are available on our project webpage.

7.3 Analysis of Stealthiness (RQ3)

Statistical Detectors. Prior research [40, 75] demonstrated that attackers can inject false data into GNSS data to induce deviations stealthily, evading both Chi-squared and CUSUM detectors [51]. Aligning with them, we define an attack as "stealthy" if injected false data does not trigger alarms.

Theoretically, Raven can be configured to minimize detection and achieve stealthiness. It can craft attacks that maintain spatio-temporal and statistical consistency with sensor data. This allows the resulting statistics from detectors to remain below their thresholds. To support this, we conducted experiments by injecting false data into GNSS readings. Similar to sensor attacks, the results of these detectors on insiders and Remote ID attackers are demonstrated in Appendix A.

To assess stealthiness, we employ the residual-based anomaly detection integrated in PX4 EKF [58]. During navigation, EKF propagates previous state estimates forward via a process model to predict subsequent GNSS measurements. Upon receiving actual GNSS data, the residual is calculated as the difference between the observed measurement and EKF's predicted measurement. If normalized residual exceeds a predefined threshold, EKF flags the measurement as anomalous.

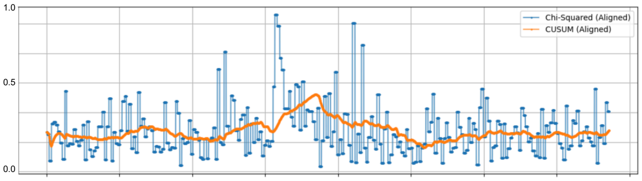

Figure 7: The real-time alarm rate of the anomaly detectors for "Navigation Delay Attack" with $n = 20$ injections.

Figure 7 demonstrates that the Raven-generated attack parameters on target drone consistently maintain test statistics below the normalized operational detection threshold (1.0). During GNSS spoofing, neither the Chi-squared detector nor the CUSUM detector (3-second window) triggered any alarms. This absence of detector alarms at standard operational thresholds demonstrates the practical stealthiness of the attacks. Raven can also adjust its attack parameters to maintain stealthiness under varying GNSS noise conditions. A real-time visualization of the detector outputs during the UAV's mission in PX4/Gazebo is available on the project website.

Performance with Heightened Detector Sensitivity. To further investigate the impracticality of detecting our Raven-generated attack parameters, which closely mimic sensor noise, we analyzed detector performance under conditions of heightened sensitivity. To assess the detectors' ability to capture these subtle attack parameters with increased sensitivity and to extract precise performance metrics under such conditions, we conducted a sensitivity analysis using moderate GNSS noise (covariance 1.0) and lower detection thresholds than the operational level of 1.0 shown in Figure 7. To calculate the optimal threshold, we utilized Youden's index [89] that maximizes the difference between TPR and FPR rates (shown in Figure 9 in Appendix). The Chi-squared detector achieved optimal performance at a threshold of 0.3596, with Precision 0.2426, Recall 0.4675, FPR 0.2014, F1 0.3194, J 0.2661. The CUSUM detector achieved optimal performance at a threshold of 0.2954, with Precision 0.3939, Recall 0.8889, FPR 0.1887, F1 0.5459, J 0.7002. These metrics reveal a critical trade-off: achieving a high TPR necessitates a high FPR. While a significantly lowered threshold can achieve a high Recall for the CUSUM detector (0.8889), this apparent improvement also results in low Precision (0.3939), indicating that the detector would frequently mark benign noise as attacks. Similarly, the Chi-squared detector, while achieving a Recall of 0.4675 under these conditions, shows an even higher FPR (0.2014) and still lower Precision (0.2426). Thus, for both detectors, increasing sensitivity (by lowering thresholds) to catch subtler attacks yields high FPR (15-20%) and very low precision (0.25), making it impractical for real-world deployment on UAVs. This impracticality is what allows the attacks to maintain their effective stealthiness.

Conclusion on Statistical Stealthiness. Our analysis leads to two key conclusions. First, Raven-generated attacks can remain undetected by both detectors. As shown in Appendix A, we obtained similar results for insider and Remote ID attack scenarios. Aligning with prior work, which conducts stealthy attacks using GNSS noise [87], `Raven` can utilize such noise in conducting attacks. Second, for noise-characteristic attacks, simply lowering the threshold is ineffective; while it might increase detections, it significantly increases false alarms. Such high FPR (0.2014) and low precision (0.2426) are impractical, as reliable anomaly detectors deployed in UAVs typically achieve FPR < 0.01 and Precision > 0.99 [3, 69].

System-level Detector. In addition to statistical detectors, system monitoring has also been used to detect GNSS spoofing. In particular, M2MON [38] protects GNSS spoofing by counting the number of ephemeris messages (n=8 in 3 minutes) in CPU's MMIO/UART path and raises an alert if it exceeds a hardcoded threshold (i.e., maximum of n=12). `Raven` can evade M2MON in three ways. (1) GNSS meaconing [49]: attacker records genuine RF signals and rebroadcasts them at higher power so each tracking channel locks onto the spoofed wave, yet the module still decodes exactly N ephemeris packets on UART. (2) Null-plus-spoof [57]: the spoofer transmits two RF signals, a phase-coherent nulling signal to erase the real satellites and a simultaneous counterfeit signal to replace them. Since M2MON only counts the GPS chip's UART output, the receiver still produces exactly N decoded packets. (3) GNSS-degraded environments: Obstructions can block satellite signals, naturally reducing the ephemeris count. As a result, the count of legitimately received ephemeris messages may fall below normal levels. This creates an opportunity for attackers to insert counterfeit signals containing fake ephemeris data without exceeding the threshold.

7.4 Root Cause Analysis (RQ4)

We analyzed all the vulnerabilities we found and categorized them into two groups, containing a total of five distinct items.

7.4.1 System-level Security Issues.

(1) High Reactivity. When the two algorithms are compared, ORCA exhibits much higher reactivity to the messages received from other robots. This makes a single false data injection attack feasible against ORCA. For instance, in the navigation delay attack, one false data injection can cause the victim robot to deviate significantly from its optimal path, choosing a longer alternative route. We attribute this to ORCA's classical optimization-based formulation, which can abruptly change the computed velocity each iteration, while GLAS, a learning-based method, results in smoother velocity changes.

(2) Imperfect Communication and Measurements. Current MRCA algorithms rely on perfect communication between the robots. Yet, in real-world scenarios, issues related to the integrity or availability of state-sharing messages can arise. In this paper, we demonstrate that FDIAs can occur due to the integration of malicious insiders in heterogeneous Internet of Drone environments, causing attacks in multi-robots.

7.4.2 Algorithmic Design Flaws

(3) Planning vs. Time Tradeoff. There is a trade-off between the planning horizon and the available time. This balance requires careful consideration in system design. Collision avoidance algorithms should plan with a longer horizon; however, this requires more time and necessitates ongoing global map maintenance. Conversely, focusing on immediate actions neglects the global map, making the robots prone to attacks.

(4) Learning-based Algorithms. For learning-based approaches such as GLAS, training in diverse environments and configurations can improve robustness against FDIAs. For instance, altering the robots' radii does not significantly impact their trajectories in GLAS, indicating that the algorithms may not guarantee collision-free navigation with different robot sizes. Similarly, changing the time step did not significantly affect the robot trajectories, suggesting that limited training data is used. For example, when we changed the radius from 0.15 to 0.5 in GLAS, the robots crashed into each other.

(5) Feasibility of Collisions. Velocity obstacle-based MRCA algorithms (e.g., ORCA) have feasible and infeasible phases. In feasible state, the algorithm guarantees a collision-free environment. However, in infeasible state, the algorithm selects the safest possible velocity, which can still result in collisions. If the attacker sends messages to introduce infeasibilities (as `Raven` discovers), they can cause the victim robot to collide with other robots and obstacles. Additionally, stopping to avoid collisions can lead to *deadlock*, where robots wait indefinitely to move out of the infeasible region.

7.5 Baseline Comparisons (RQ5)

Naive Attacks. We compare `Raven` to naive attacks in two different environments. The naive approach selects a random time between $[0, \text{mission\_time}]$ and chooses a random position displacement within the same constraints as `Raven`. We ran each method ten times and present their success rates in Table 3. We tested both methods with 1, 5, and 10 FDIs. Random attacks never produced shepherding attacks and only achieved a 10% collision rate with five false messages. In contrast, `Raven` achieved a 100% success rate in discovering correct attack parameters for both attack categories.

Comparison with SF. The most relevant work on finding vulnerabilities in AMRs are SwarmFuzz (SF) [87] and SwarmFlawFinder (SFF) [34]. Among them, SFF cannot perform remote attacks because it requires an external robot to interact with the victim physically. In contrast, SF performs GPS spoofing attacks by manipulating targets to cause collisions.

Table 3: A comparison of Raven with baseline methods.

	Shepherding Attacks		Collision Attacks	
Method	**#Injections**	**Attack Success %**	**#Injections**	**Attack Success %**
Benign	-	0	-	0
Naive Attacks	1/5/10	0/0/0	1/5/10	0/10/20
Swarmfuzz [87]	1/5/10	-/-/-	1/5/10	0/0/0
Raven	1/5/10	100/100/100	1/5/10	100/100/100

We compared Raven with SF both qualitatively and quantitatively. To summarize SF's methodology, SF utilizes graph theory to select victim-target pairs and employs gradient descent to determine attack parameters. Its methodology focuses on one objective (robot-obstacle collisions). We integrated SF into our environment to test on ORCA and GLAS.

We ran their methodology with three robots in two different environments. To align with their methodology, we conducted experiments on each obstacle individually, ensuring only one was present in the environment at a time. However, SF could not detect any robot-obstacle collisions. We investigated potential causes. We found that SF requires the target and victim to pass on different sides of the obstacle. This requirement prevented SF from identifying collisions that Raven can detect. Next, we modified the environment to have one robot pass left of an obstacle and the other two pass right. However, SF could not find the correct attack parameters. We analyzed the root causes and identified three main restrictions: (1) conducting only horizontal spoofing, (2) conducting only constant spoofing (i.e., always the same displacement), and (3) the target robot must align with the obstacle along the x-axis.

7.6 Time Efficiency (RQ6)

We measured the time required to find the attack parameters for each attacker goal. Table 2 "Attack Plan Time" column shows the min/median/max time values. For instance, the attack plan time for a herding attack on ORCA is 1.97 s / 2.26 s / 2.53 s. Although the configurations, such as maximum speed, were proportionally set to have similar conditions in both algorithms, it took more than three times longer on average to find the attacks on GLAS than on ORCA. For example, it takes 8 min 4 s / 10 min 2 s / 14 min 8 s on GLAS to find robot-obstacle collisions. Thus, Raven requires more runs and iterations to find the attack parameters against GLAS.

8 Discussion and Limitations

Chain-of-Reactions in Large Multi-Robot Systems. In large multi-robot systems, manipulating a robot's position initiates cascading indirect effects throughout the system. From an attacker's standpoint, while the inherent density of large multi-robot systems complicates maintaining a collision-free state, executing targeted attacks (e.g., herding) becomes more challenging. Such targeted attacks necessitate a higher number of FDIs and time for Raven to find effective parameters.

Vision-based Detect-and-Avoid. One way to mitigate our attacks could be leveraging vision sensors (e.g., camera or LiDAR) to confirm the validity of the received data. However, recent work [26, 85] showed that image classifiers can be misled by adding small adversarial noises to the input (e.g., a street sign image). Therefore, our attacks can be combined with adversarial noises to evade vision-based attack detection.

CA Algorithms in Autopilot Frameworks. We analyzed collision-avoidance (CA) algorithms in widely used open-source SITL Autopilot frameworks (Ardupilot [7], PX4 [58], Paparazzi [55]). However, current implementations do not support collective multi-robot collision avoidance as they do not incorporate other robots' positions or velocities into the ego vehicle's next waypoint calculation. Instead, they support single-robot collision avoidance. To demonstrate whether future extensions of these algorithms are vulnerable to our attacks, we conducted an experiment in Ardupilot. In Appendix A, we demonstrate that the Bendyruler algorithm [52] in Ardupilot remains vulnerable to deadlock attacks in a scenario with static obstacles and geofences. Thus, we expect that extending these algorithms to multi-robots would yield similar attack success rates. Furthermore, each robot may operate with an individual CA algorithm. However, even in such environments, FDIAs can achieve similar success rates. Beyond the five attacks discussed in this paper, "livelock" attacks are also possible, where two robots repeatedly move in the same direction due to independently operating CA algorithms.

Attack Recovery Frameworks. Raven's stealthiness against standard anomaly detectors (in Section 7.3) directly undermines current attack recovery frameworks, as the detection stages are crucial for activating contemporary recovery systems [19, 21]. DeLorean [21] employs a diagnosis-guided recovery approach against Sensor Deception Attacks (SDAs). It identifies targeted sensors through causal analysis and reconstructs states selectively using historical data. Similarly, SpecGuard [19] adopts a specification-aware recovery method leveraging Deep Reinforcement Learning (Deep-RL) to maintain mission compliance during attacks. Its reactive control variant, like DeLorean, activates only upon external detection alerts. These mechanisms critically depend on accurate and timely alerts from anomaly detectors. However, as demonstrated in Section 7.3, detectors prone to false alarms either fail to trigger recovery or cause inappropriate recovery initiations. By effectively evading or delaying these alerts, Raven's attacks can significantly compromise the reliability and effectiveness of existing recovery frameworks.

Potential Mitigations to Root Causes of Vulnerabilities. To address the identified root causes, we propose the following improvements: (1) *Learning-based Algorithms*: Robustness against manipulated data in learning-based algorithms can be enhanced through training with diverse datasets and employing adversarial training techniques. (2) *Feasibility of Collisions*: Defending against attacks that induce infeasible states

Table 4: A comparison of Raven with vulnerability discovery systems in autonomous multi robots.

System	Input to Target Vehicle	Remote Attack Capability	Multiple Attackers Support	Multiple Victims Support	Attack Optimization	Attack Stealthiness	Deadlock Attacks
SFF [34]	Continuous Maneuver	No (External Attack Drone)	✓	✓	N/A	×	×
SF [87]	Constant Spoofing	✓	×	×	✓	×	×
Raven	Intermittent Spoofing	✓	✓	✓	✓	✓	✓

† SFF: SwarmFlawFinder, SF: SwarmFuzz.

or unavoidable collisions can involve handling such states separately, for instance, via priority-based movement, or by setting conservative safety configurations. (3) *High-Reactivity*: Counteracting vulnerabilities arising from high reactivity to false messages can be achieved by applying temporal filtering to smooth out sudden, potentially malicious, changes in incoming data. (4) *Imperfect Communication*: Implementing message authentication for internal networks against intruders, employing encryption and authentication for broadcast systems like Remote-ID, and utilizing techniques such as Multilateration (MLAT) for verifying GNSS signals can mitigate false data injection and spoofing [37, 77]. (5) *Planning vs. Time Tradeoff*: Improving the quality and safety of collision avoidance decisions can involve planning over a longer time horizon while maintaining execution at shorter periods.

9 Related Work

Vulnerability Discovery in Robotic Vehicles. Prior research has investigated the vulnerabilities of robotic vehicles. Recent work [41] explored bugs by fuzzing robots' control software. Another study [16] discovered bugs in the safety checks of robotic vehicles. Another work [20] proposes three types of attacks (false data injection, artificial delay, and switch-mode attacks) targeting robotic vehicles. However, these studies focus on a single robot and do not address bugs/vulnerabilities in multi-robot navigation. In the autonomous vehicle (AV) domain, prior work has discovered attacks using maneuvers [44,66,73]. However, the AV domain differs from mobile robots due to different objectives, such as lane keeping, and different control systems with varying degrees of freedom.

Similarly, in vehicle platooning domain, prior studies [1,18] have examined FDIAs, with practical countermeasures [11, 50]. However, FDIAs in vehicle platooning differentiate from our work in two ways: (1) vehicle platooning operates as a single unit along fixed lanes. Thus, vehicles have limited degrees of freedom and maintain a rigid formation, and (2) vehicle platooning follows a leader-follower architecture. The leader sets the trajectory and speed for the platoon, where compromising a leader affects the entire platoon. Instead, MRCA algorithms we focus on operate without enforcing a rigid, lane-bound formation and are fully decentralized.

Table 4 compares Raven with previous frameworks for finding vulnerabilities of multi-robot systems. SwarmFlawFinder [34] requires an external drone for its attack methodology [13]. It focuses on swarm algorithms that prioritize formation over collision avoidance, using an additional formation force along with goal and collision avoidance forces. Furthermore, it limits its search space to only four discrete actions. Unlike SwarmFlawFinder, Raven focuses on MRCA algorithms and does not require close proximity to robots. Additionally, we do not limit the search space or constrain our mutation strategy to discrete actions.

SwarmFuzz [87] focuses on swarm control algorithms and finds propagation vulnerabilities. Its methodology focuses on horizontal spoofing (i.e., left or right), which limits the search space for finding diverse and unique attacks. SwarmFuzz also attempts to discover stealthy attacks but employs constant spoofing, meaning the spoofing continues until the victim robot is hit or the framework times out. This approach reduces stealthiness and can be detected by window-based detectors [60]. In contrast, Raven investigates the MRCA algorithms and introduces a framework without limiting the search space. Moreover, Raven considers ripple effects by checking all direct and indirect interactions among robots to achieve five distinct attack goals. Unlike SwarmFuzz, Raven also minimizes the number of injections through discrete spoofing. This method allows the attacker to choose different timestamps for conducting attacks, thereby minimizing the number of FDIAs and reducing the risk of detection.

Optimal and Stealthy Attack Planning in Multi-Robots. Recent work [39] has focused on stealthy attacks on UAVs, primarily targeting perception modules. In contrast, our research targets multi-robot navigation algorithms. Another study [83] introduced the concept of physical masquerade attacks in multi-robot systems, assuming one robot is compromised. While their work focuses on multi-robot pathfinding and includes observation planning for attack detection, it does not address the optimization and minimization of attack parameters for stealthy attacks. To the best of our knowledge, Raven is the first to perform optimal attacks by minimizing the number of false data injections to achieve shepherding and collision attacks in multi-robot systems. Our approach ensures that attacks are both effective and difficult to detect.

10 Conclusion

We introduced Raven, a tool that identifies effective and stealthy attacks on MRCA algorithms. Specifically, we identified and analyzed vulnerabilities in state-of-the-art MRCA algorithms. We found these algorithms vulnerable to FDIAs, leading to shepherding and collision attacks. Our experiments under various configurations showed the effectiveness of injections in disrupting multi-robot missions. The inherent complexity of cooperation in these systems highlights the need for robust MRCA algorithms to address these vulnerabilities.

11 Acknowledgments

We appreciate the valuable feedback and suggestions from our shepherd and anonymous reviewers. We also thank Baskın Şenbaşlar for helpful discussions during idea exploration. This research was partially funded by the National Science Foundation (NSF) through grants CNS-2144645 and IIS-2229876. The findings, conclusions, and recommendations presented in this paper are solely those of the authors and do not necessarily represent the views of the NSF.

12 Ethics Considerations

Ethical Considerations. This paper proposes new attacks against MRCA algorithms, which can cause shepherding and collision attacks. Discovering these attacks is vital to improving the security of MRCA against FDIAs. AMR developers can use Raven to identify vulnerabilities in their MRCA algorithms. They can then revisit their assumptions or patch their algorithms to mitigate the attacks. Since Raven uses temporal logic, developers can also define their formulas and test their algorithms. To further mitigate the negative consequences of publishing new attacks, we propose potential countermeasure techniques. To this end, we disclosed our findings to the ORCA and GLAS developers. Our findings pave the way for future research focused on enhancing the robustness and resilience of MRCA algorithms in adversarial conditions.

Stakeholders. The semantic vulnerabilities identified in this study diverge from conventional software bugs. Instead of merely uncovering typical software bugs, our approach reveals mission deviations in multi-robots that pose potential risks to human safety, robotic operations, and the environment. Consequently, we have communicated our findings to industry stakeholders and agencies. Specifically, we have shared our report with ten multi-robot companies or agencies: Amazon Robotics (Prime Air), Wing, Waymo, Cruise, Starship Technologies, Nuro, Alibaba, Cybersecurity and Infrastructure Security Agency, Federal Aviation Administration, and European Union Aviation Safety Agency.

Mitigating Potential Harm. We did not conduct indoor or outdoor GNSS spoofing attacks. We demonstrated false data injections from insiders in our real-world experiments. We conducted such experiments in a lab environment to mitigate any potential harm to humans or the environment.

13 Open Science

We have provided the artifacts of our study online [88]. We have made the implementations of attacks on ORCA and GLAS, the tested environments, supplementary experiments, and the visualization library publicly available.

References

[1] Ahmed Abdo, Sakib Md Bin Malek, Zhiyun Qian, Qi Zhu, Matthew Barth, and Nael Abu-Ghazaleh. Application level attacks on connected vehicle protocols. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2019.

[2] Laith Abualigah, Ali Diabat, Putra Sumari, and Amir H Gandomi. Applications, deployments, and integration of internet of drones (iod): a review. *IEEE Sensors Journal*, 2021.

[3] Khattab M Ali Alheeti, Fawaz Khaled Alarfaj, Mohammed Alreshoodi, Naif Almusallam, and Duaa Al Dosary. A hybrid security system for drones based on icmetric technology. *Plos one*, 2023.

[4] Javier Alonso-Mora, Andreas Breitenmoser, Paul Beardsley, and Roland Siegwart. Reciprocal collision avoidance for multiple car-like robots. In *IEEE International Conference on Robotics and Automation*, 2012.

[5] Amazon patented a fantastical floating airship warehouse for its delivery drones. https://www.theverge.com/2016/12/29/14114190/amazon-patent-drone-airship-delivery, 2016. [Online; accessed 1-December-2023].

[6] Amazon's 100 drone deliveries puts prime air far behind alphabet's wing and walmart partner zipline. http://tinyurl.com/3yrzby8b, 2023. [Online; accessed 1-December-2023].

[7] Ardupilot. https://ardupilot.org, 2024. [Online; accessed 1-June-2024].

[8] Syeda Nazia Ashraf, Selvakumar Manickam, Syed Saood Zia, Abdul Ahad Abro, Muath Obaidat, Mueen Uddin, Maha Abdelhaq, and Raed Alsaqour. Iot empowered smart cybersecurity framework for intrusion detection in internet of drones. *Scientific Reports*, 2023.

[9] Automatic dependent surveillance-broadcast (ads-b). https://www.faa.gov/air_traffic/technology/adsb, 2023. [Online; accessed 01-January-2025].

[10] Lailla MS Bine, Azzedine Boukerche, Linnyer B Ruiz, and Antonio AF Loureiro. Leveraging urban computing with the internet of drones. *IEEE Internet of Things Magazine*, 2022.

[11] Roghieh A Biroon, Zoleikha Abdollahi Biron, and Pierluigi Pisu. False data injection attack in a platoon of cacc: Real-time detection and isolation with a pde approach. *IEEE transactions on intelligent transportation systems*, 2021.

[12] Browse public log files. https://review.px4.io/browse, 2024. [Online; accessed 1-January-2025].

[13] Matthias R Brust, Grégoire Danoy, Pascal Bouvry, Dren Gashi, Himadri Pathak, and Mike P Gonçalves. Defending against intrusion of malicious uavs with networked uav defense swarms. In *IEEE Local Computer Networks workshops (LCN workshops)*, 2017.

[14] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z Morley Mao. Adversarial sensor attack on lidar-based perception in autonomous driving. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[15] Ming Chen, Jianyi Liu, Jinlong Pang, Zhiqiang Jian, Pei Chen, and Xinhu Zheng. Multi-risk aware trajectory planning for car-like robot in highly dynamic environments. In *IEEE Intelligent Transportation Systems (ITSC)*, 2023.

[16] Hongjun Choi, Sayali Kate, Yousra Aafer, Xiangyu Zhang, and Dongyan Xu. Cyber-physical inconsistency vulnerability identification for safety checks in robotic vehicles. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[17] Andrei Costin, Aurélien Francillon, et al. Ghost in the air (traffic): On insecurity of ads-b protocol and practical attacks on ads-b devices. *Black Hat USA*, 2012.

[18] Soodeh Dadras, Ryan M Gerdes, and Rajnikant Sharma. Vehicular platooning in an adversarial environment. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[19] Pritam Dash, Ethan Chan, and Karthik Pattabiraman. Specguard: Specification aware recovery for robotic autonomous vehicles from physical attacks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[20] Pritam Dash, Mehdi Karimibiuki, and Karthik Pattabiraman. Out of control: stealthy attacks against robotic vehicles protected by control-based techniques. In *Annual Computer Security Applications Conference*, 2019.

[21] Pritam Dash, Guanpeng Li, Mehdi Karimibiuki, and Karthik Pattabiraman. Diagnosis-guided attack recovery for securing robotic vehicles from sensor deception attacks. In *ACM ASIA Conference on Computer and Communications Security*, 2024.

[22] Abdelouahid Derhab, Omar Cheikhrouhou, Azza Allouch, Anis Koubaa, Basit Qureshi, Mohamed Amine Ferrag, Leandros Maglaras, and Farrukh Aslam Khan. Internet of drones security: Taxonomies, open issues, and future directions. *Vehicular Communications*, 2023.

[23] Vishal Dey, Vikramkumar Pudi, Anupam Chattopadhyay, and Yuval Elovici. Security vulnerabilities of unmanned aerial vehicles and countermeasures: An experimental study. In *VLSID*, 2018.

[24] Distributed optimal reciprocal collision avoidance. `https://github.com/lis-epfl/swarmlab`, 2019. [Online; accessed 1-June-2024].

[25] Drone wars: Developments in drone swarm technology. `https://tinyurl.com/46tvc7v9`, 2025. [Online; accessed 21-January-2025].

[26] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. *arXiv preprint arXiv:1707.08945*, 2017.

[27] Federal aviation administration. `%https://www.faa.gov/`, 2024. [Online; accessed 01-January-2025].

[28] gazebo. `https://gazebosim.org/home`, 2024. [Online; accessed 1-June-2024].

[29] Georgia tech's new coordinated drone swarm delivery system. `https://consortiq.com/uas-resources/drone-swarm-delivery-system`, 2023. [Online; accessed 1-December-2023].

[30] Mirmojtaba Gharibi, Raouf Boutaba, and Steven L Waslander. Internet of drones. *IEEE Access*, 2016.

[31] User guidelines for single base real time gnss positioning. `https://www.ngs.noaa.gov/PUBS_LIB/NGSRealTimeUserGuidelines.v2.1.pdf`, 2024. [Online; accessed 1-June-2024].

[32] Dynamic harmonic notch filters. `https://ardupilot.org/copter/docs/common-imu-notch-filtering.html`, 2024. [Online; accessed 1-June-2024].

[33] Bart Hermans and Luc Gommans. Targeted gps spoofing. *Research Project Report*, 2018.

[34] Chijung Jung, Ali Ahad, Yuseok Jeon, and Yonghwi Kwon. Swarmflawfinder: Discovering and exploiting logic flaws of swarm algorithms. In *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[35] Homayun Kabir, Mau-Luen Tham, and Yoong Choon Chang. Internet of robotic things for mobile robots: concepts, technologies, challenges, applications, and future directions. *Digital Communications and Networks*, 2023.

[36] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[37] Colin Kelsey, David Laverty, and John O'Raw. Multilateration method for locating gnss spoofing attacks affecting substation clocks. In *2022 33rd Irish Signals and Systems Conference (ISSC)*, 2022.

[38] Arslan Khan, Hyungsub Kim, Byoungyoung Lee, Dongyan Xu, Antonio Bianchi, and Dave Jing Tian. M2MON: Building an MMIO-based security reference monitor for unmanned vehicles. In *USENIX Security*, 2021.

[39] Amir Khazraei, Haocheng Meng, and Miroslav Pajic. Stealthy perception-based attacks on unmanned aerial vehicles. *arXiv preprint arXiv:2303.02112*, 2023.

[40] Amir Khazraei, Haocheng Meng, and Miroslav Pajic. Blackbox stealthy gps attacks on unmanned aerial vehicles. *arXiv preprint arXiv:2409.11405*, 2024.

[41] Hyungsub Kim, Muslum Ozgur Ozmen, Antonio Bianchi, Z Berkay Celik, and Dongyan Xu. Pgfuzz: Policy-guided fuzzing for robotic vehicles. In *NDSS*, 2021.

[42] Hyungsub Kim, Muslum Ozgur Ozmen, Z Berkay Celik, Antonio Bianchi, and Dongyan Xu. Pgpatch: Policy-guided logic bug patching for robotic vehicles. In *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[43] Hyungsub Kim, Muslum Ozgur Ozmen, Z Berkay Celik, Antonio Bianchi, and Dongyan Xu. Patchverif: Discovering faulty patches in robotic vehicles. In *USENIX Security*, 2023.

[44] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. Av-fuzzer: Finding safety violations in autonomous driving systems. In *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2020.

[45] Low-pass filter. `https://docs.px4.io/main/en/config_mc/filter_tuning.html`, 2024. [Online; accessed 1-June-2024].

[46] Making search and rescue drone swarms a reality. https://eng.vt.edu/magazine/stories/fall-2021/drone-swarms.html, 2021. [Online; accessed 1-December-2023].

[47] Matlab. https://www.mathworks.com/products/matlab.html, 2024. [Online; accessed 1-June-2024].

[48] Mavlink. https://mavlink.io/en/, 2024. [Online; accessed 8-December-2024].

[49] Lianxiao Meng, Lin Yang, Wu Yang, and Long Zhang. A survey of gnss spoofing and anti-spoofing technology. *Remote sensing*, 2022.

[50] Eman Mousavinejad, Fuwen Yang, Qing-Long Han, Xiaohua Ge, and Ljubo Vlacic. Distributed cyber attacks detection and recovery mechanism for vehicle platooning. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[51] Carlos Murguia and Justin Ruths. Cusum and chi-squared attack detection of compromised sensors. In *IEEE Conference on Control Applications (CCA)*, 2016.

[52] Object avoidance using dijkstra's with bendyruler. https://ardupilot.org/copter/docs/common-oa-dijkstrabendyruler.html/, 2024. [Online; accessed 1-January-2025].

[53] Reza Olfati-Saber and Richard M Murray. Distributed cooperative control of multiple vehicle formations using structural potential functions. *IFAC Proceedings Volumes*, 2002.

[54] Kartik A Pant, Li-Yu Lin, Jaehyeok Kim, Worawis Sribunma, James M Goppert, and Inseok Hwang. Mixed-sense: A mixed reality sensor emulation framework for test and evaluation of uavs against false data injection attacks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

[55] Paparazzi uas. https://github.com/paparazzi/paparazzi/, 2024. [Online; accessed 1-June-2024].

[56] Amazon prime air prepares for drone deliveries. https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries, 2022. [Online; accessed 25-December-2023].

[57] Mark L Psiaki and Todd E Humphreys. Gnss spoofing and detection. *Proceedings of the IEEE*, 2016.

[58] Px4. https://px4.io, 2024. [Online; accessed 1-June-2024].

[59] Jorge Pena Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *IEEE Access*, 2020.

[60] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. SAVIOR: Securing autonomous vehicles with robust physical invariants. In *USENIX Security*, 2020.

[61] Remote identification of drones. https://www.faa.gov/uas/getting_started/remote_id, 2024. [Online; accessed 01-January-2025].

[62] Remote identification of unmanned aircraft. https://www.ecfr.gov/current/title-14/chapter-I/subchapter-F/part-89, 2024. [Online; accessed 01-January-2025].

[63] Benjamin Riviere, Wolfgang Hönig, Yisong Yue, and Soon-Jo Chung. Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *IEEE Robotics and Automation Letters*, 2020.

[64] Arnau Romero, Carmen Delgado, Lanfranco Zanzi, Raúl Suárez, and Xavier Costa-Pérez. Cellular-enabled collaborative robots planning and operations for search-and-rescue scenarios. *arXiv preprint arXiv:2403.09177*, 2024.

[65] ros. https://www.ros.org/, 2024. [Online; accessed 1-June-2024].

[66] Ivan F Salgado, Nicanor Quijano, Daniel J Fremont, and Alvaro A Cardenas. Fuzzing malicious driving behavior to find vulnerabilities in collision avoidance systems. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2022.

[67] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 2019.

[68] Harshad Sathaye, Martin Strohmeier, Vincent Lenders, and Aanjhan Ranganathan. An experimental study of GPS spoofing and takeover attacks on UAVs. In *USENIX Security*, 2022.

[69] Savio Sciancalepore, Omar Adel Ibrahim, Gabriele Oligeri, and Roberto Di Pietro. Picking a needle in a haystack: Detecting drones via network traffic analysis. *arXiv preprint arXiv:1901.03535*, 2019.

[70] Seeing through forest with drone swarms. https://ecoevocommunity.nature.com/posts/seeing-through-forest-with-drone-swarms, 2022. [Online; accessed 1-December-2023].

[71] Baskın Şenbaşlar, Wolfgang Hönig, and Nora Ayanian. Rlss: real-time, decentralized, cooperative, networkless multi-robot trajectory planning using linear spatial separations. *Autonomous Robots*, 2023.

[72] Baskın Şenbaşlar, Pilar Luiz, Wolfgang Hönig, and Gaurav S Sukhatme. Mrnav: Multi-robot aware planning and control stack for collision and deadlock-free navigation in cluttered environments. *arXiv preprint arXiv:2308.13499*, 2023.

[73] Ruoyu Song, Muslum Ozgur Ozmen, Hyungsub Kim, Raymond Muller, Z Berkay Celik, and Antonio Bianchi. Discovering adversarial driving maneuvers against autonomous vehicles. In *USENIX Security*, 2023.

[74] Spoofing drone locations by manipulating remote id protocols and communications. https://tinyurl.com/56w5wb95, 2024. [Online; accessed 25-December-2023].

[75] Jie Su, Jianping He, Peng Cheng, and Jiming Chen. A stealthy gps spoofing strategy for manipulating the trajectory of an unmanned aerial vehicle. *IFAC-PapersOnLine*, 49(22):291–296, 2016.

[76] swarmlab. https://github.com/lis-epfl/swarmlab, 2024. [Online; accessed 1-June-2024].

[77] Pietro Tedeschi, Savio Sciancalepore, and Roberto Di Pietro. Arid: Anonymous remote identification of unmanned aerial vehicles. In *Annual Computer Security Applications Conference*, 2021.

[78] Quinn Thibeault, Jacob Anderson, Aniruddh Chandratre, Giulia Pedrielli, and Georgios Fainekos. Psy-taliro: A python toolbox for search-based test generation for cyber-physical systems. In *FMICS*, 2021.

[79] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful gps spoofing attacks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2011.

[80] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *International Symposium on Robotics Research (ISRR)*, 2011.

[81] Gábor Vásárhelyi, Csaba Virágh, Gergő Somorjai, Tamás Nepusz, Agoston E Eiben, and Tamás Vicsek. Optimized flocking of autonomous drones in confined environments. *Science Robotics*, 2018.

[82] Li Wang, Aaron D Ames, and Magnus Egerstedt. Safety barrier certificates for collisions-free multirobot systems. *IEEE Transactions on Robotics*, 2017.

[83] Kacper Wardega, Roberto Tron, and Wenchao Li. Resilience of multi-robot systems to physical masquerade attacks. In *IEEE S&P Workshops (SPW)*, 2019.

[84] Xrce. https://www.eprosima.com/index.php/resources-all/whitepapers/xrce, 2024. [Online; accessed 1-June-2024].

[85] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world. In *ECCV*, 2020.

[86] Zhen Yang, Jun Ying, Junjie Shen, Yiheng Feng, Qi Alfred Chen, Z Morley Mao, and Henry X Liu. Anomaly detection against gps spoofing attacks on connected and autonomous vehicles using learning from demonstration. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[87] Yingao Elaine Yao, Pritam Dash, and Karthik Pattabiraman. Swarmfuzz: Discovering gps spoofing attacks in drone swarms. In *IEEE/IFIP DSN*, 2023.

[88] Doguhan Yeke, Kartik Anand Pant, M. Ozgur Ozmen, Hyungsub Kim, James M. Goppert, Inseok Hwang, Antonio Bianchi, and Z. Berkay Celik. Automated Discovery of Semantic Attacks in Multi-Robot Navigation Systems - Research Artifacts. https://doi.org/10.6084/m9.figshare.29260229, 2025. [Online; accessed 8-June-2025].

[89] William J Youden. Index for rating diagnostic tests. *Cancer*, 1950.

[90] Tian-Yu Zhang, Dan Ye, and Guang-Hong Yang. Ripple effect of cooperative attacks in multi-agent systems: Results on minimum attack targets. *Automatica*, 2024.

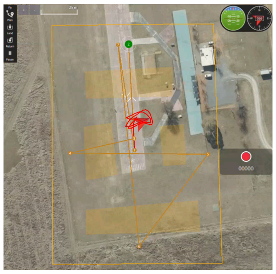

Figure 8: A deadlock illustration on BendyRuler/Ardupilot.

Table 5: Raven's performance on all target-victim pairs for navigation delay attack with n={5,10} FDIs.

Target	Victim	Success Rate
0	1	Yes (Suboptimal, n=10)
0	2	No
1	0	Yes (Optimal, n=5)
1	2	Yes (Optimal, n=5)
2	0	Yes (Suboptimal, n=10)
2	1	No

A Evaluation Details

MRCA Algorithm Selection Process. We conducted a systematic literature review following a three-stage filtering process. First, we searched for papers published between 2010 and 2024 using the keywords "multi-robot collision avoidance", "multi-agent path finding", "multi-robot motion planning", and "n-body collision avoidance". We confined the search to top-tier robotics venues (ICRA, IROS, RSS, RAL, IJRR, T-RO, and Robotics Research) and retrieved 512 relevant publications. Second, we reduced it to: (1) prioritize collision avoidance over formation maintenance, (2) offer an open-source implementation, (3) demonstrate performance on real robots. Third, we divided the remaining papers into two categories—classical and learning-based. From each category, we identified the top three papers ranked by their impact. Based on citation metrics and comparative performance from existing literature [63], we chose ORCA from the classical approaches and GLAS from the learning-based algorithms.

Supporting other MRCA Algorithms. `Raven` does not depend on the internal details of velocity-based MRCA algorithms, allowing it to be extended for any MRCA algorithm. Specifically, `Raven` wraps four existing methods from the MRCA algorithm codebase: `init_MRCA()` initializes an MRCA instance with parameters such as initial positions, goal positions, and obstacle positions; `set_positions()` injects false data to simulate FDIA; `get_positions()` retrieves updated positions at the next timestep; and `calculate_next_velocities()` runs the MRCA algorithm at each timestep. Thus, `Raven` neither introduces new functions nor modifies internal MRCA components; it employs these four wrappers to integrate with any MRCA algorithm.

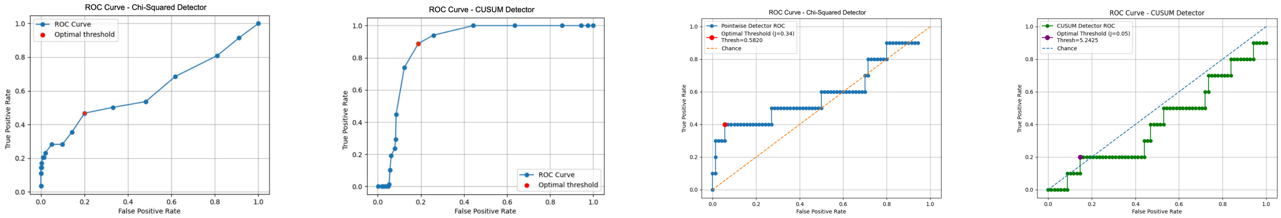

Figure 9: ROC curves for lowered thresholds for an effective GNSS spoofing and insider/Remote ID attack scenario.

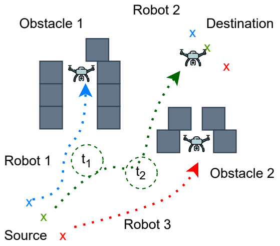

Figure 10: Attacking multiple victim robots on ORCA.

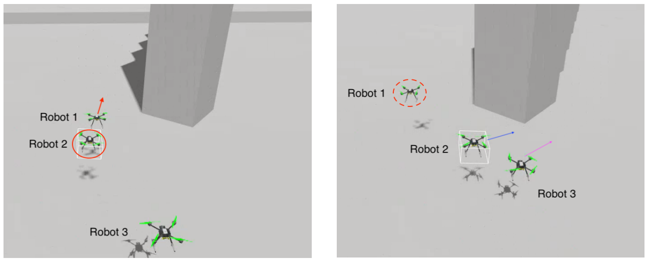

Figure 12: Deadlock attack sequence. (a) The initial state, where robots move toward their targets. (b) The attacker intercepts the victim, forcing it onto a new path where it becomes permanently stuck due to its collision avoidance constraints.

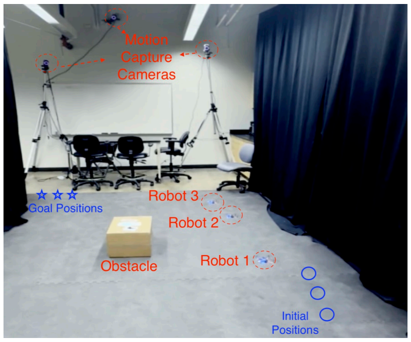

Figure 11: Real-world experiment setup.

Supporting New Attacks. Our literature review revealed five primary attacks in multi-robot systems, constituting fundamental adversarial strategies. However, Raven is easily extendable for developers to add new attacks by simply defining them as STL-formulas, similar to those in Table 1.

Attack Reproducibility with Imprecise Spoofing. We tested the reproducibility with imprecise spoofing. Initially, we identified a "navigation delay attack" using one false data injection with parameters: time = 5.9 s, x = -0.38, y = 3.39. We then altered the position displacement by sampling from [-1,+1] and ran the experiment 5 times. The attack was successful in 3 of 5 trials. In failed cases, the position displacement was insufficient to induce a deadlock position for the victim robot.

Porting Raven to Commercial MRCA Algorithms. Commercial MRCA algorithms are not available for public use, so our experiments utilize publicly accessible MRCA algorithms. Our analysis focused on two MRCA algorithms, ORCA and GLAS, as representative cases. Raven does not rely on any specific design features of these algorithms. Therefore, Raven can easily be extended to the commercial MRCA algorithms.

Attacks on Open-Source Autopilot Frameworks. We conducted an experiment using Bendyruler [52], the object avoidance algorithm in Ardupilot [7], with OA_TYPE=1. Our experiment shows that the current version of the algorithm is vulnerable to deadlock attacks. We set up the environment by introducing exclusion geofences and real obstacles. In our SITL setup, the robot detects obstacles via LiDAR. Figure 8 displays a QGroundControl map where the robot becomes trapped by an obstacle. The robot exhibits negligible movement and cannot change its position significantly for more than 10 seconds. These results show that object avoidance algorithms in popular controllers may fail under FDIAs.

Different Environments. We conduct experiments in different environments (Env1, Env2, Env3, and Env4). Each environment contains distinct obstacle positions and shapes to capture various testing conditions. The start and goal positions of robots also differ. Although three robots are shown for illustration, Raven can operate with any number of robots.

Case Study 3 (Fig. 10) - *Attacking Multiple Victims.* In addition to attacking a single victim, Raven can also be extended to attack multiple victims. In this case study, we demonstrate how one target robot can immobilize two victim robots through a deadlock attack demonstrated in the Python simulator. We conducted tests using n=10 FDIs. Figure 10 shows the attack's effect on ORCA. The target robot (Robot 2) successfully disrupted two victim robots, placing them in separate deadlock positions. Initially, the target robot navigates to the left, pushing victim Robot 1 into Obstacle 1 at t_1. Then, at t_2, the target robot moves right, causing victim Robot 3 to become trapped in a deadlock. The victims (Robot 1 and Robot 3) consider moving towards the target, but MRCA algorithm prevents a collision, resulting in a deadlock position.

Deadlock Attack Illustration in PX4/Gazebo Simulation. Figure 12 demonstrates the attack in a PX4/Gazebo simulation. The victim robot (Robot 1) steers left to avoid a collision with the target robot (Robot 2). From this new position, Robot 1 attempts to move toward its goal but is obstructed by Robot 2. The robot eventually becomes stuck because its collision avoidance constraints prevent it from navigating to the goal.

Statistical Detectors on Insiders and Remote ID Attacks. Insiders can precisely inject false data into messages exchanged between robots. Similarly, exploiting the absence of encryption and authentication in FAA Remote ID standards [61], Remote ID attackers can craft equally precise false data by specifying target robot IDs and exact false coordinates. Therefore, from the perspective of a victim evaluating incoming position messages, the core anomaly introduced by either attack type appears indistinguishable to the victim.

We adapted Chi-squared and CUSUM detectors for insider and Remote ID attackers. Initially designed for GNSS anomaly detection, these detectors can also handle anomalies from incoming position messages. For these attack vectors, each robot evaluates incoming position messages, not raw GNSS coordinates. Each robot records a received position as its "previous state" to predict a neighbor's subsequent location. From the next message, the residual (i.e., difference between predicted and actual positions) is computed. Residuals are flagged as anomalous if they exceed predefined thresholds.

To illustrate that Raven can generate stealthy messages, in a three-robot navigation delay attack, an adversary conducts n=11 FDIs. Figure 13 shows incoming data from the "target robot" conducting FDIAs, where each robot uses both Chi-squared and CUSUM detectors on the incoming data. The adversary carefully crafted false messages (marked by red lines in Figure 13), causing small deviations on the victim and resulting in residuals below detection thresholds (i.e., no flags are triggered at any time instance for both detectors). To calculate the optimal threshold, we utilized Youden's index [89], which maximizes the difference between the TPR and FPR rates. The Chi-squared detector achieved optimal performance at a threshold of 0.5819, with Precision 0.50, Recall 0.41, FPR 0.05, F1 0.44, J 0.34. The CUSUM detector achieved optimal performance at a threshold of 0.4368, with Precision 0.16, Recall 0.20, FPR 0.14, F1 0.18, J 0.05. Compared to sensor attacks, detection rates are slightly lower for insider/Remote ID attacks. There are three main reasons. (1) EKF applies additional filtering mechanisms to exclude anomalies in sensor data. (2) EKF uses nonlinear modeling for state prediction, while insider modeling relies on a simplified version. (3) Differences in key parameters (e.g., message frequency and noise levels) vary the resulting data. Consequently, our experimental results reveal two insights: (1) Slight position displacements to evade the Chi-squared detector necessitate more FDIs for a successful and stealthy attack. (2) Intermittent attacks (i.e., discrete attacks) can help circumvent window-based CUSUM detectors, as such attacks do not continuously inject false data.

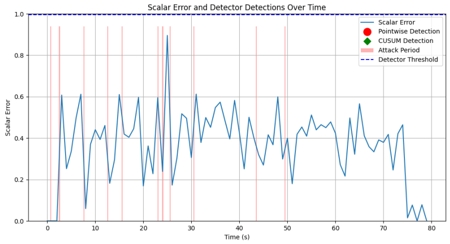

Figure 13: Detector output for an insider/Remote ID attack.

B Simulator Details

We explored open-source contributions and discovered the Distributed Optimal Reciprocal Collision Avoidance (d-ORCA) [24] framework. Although d-ORCA implements ORCA [80] and deploys it to multi-robots, its software components are outdated. It operates on Ubuntu 16, Robot Operating System (ROS) Kinetic, and Gazebo 7.13. However, ROS [65] has transitioned to ROS2, and Gazebo [28] is now in version 11, each version bringing significant improvements such as enhanced sensor plugins and real-time communication that affect the performance and precision of the experiments. Furthermore, d-ORCA does not support any other MRCA algorithms. We tried upgrading its components, yet it was infeasible due to its design for specific software versions.

We developed our simulator, which comprises five key components. First, Gazebo [28] creates the world and spawns the models, including robots and obstacles. Gazebo then sends raw sensor data, such as GNSS and accelerometer readings, to robots via its sensor plugins. To enable communication between components, XRCE [84] creates real-time low-level channels. Next, PX4 [58] subscribes to these channels to receive sensor measurements. Note that we deploy separate PX4 Autopilots to each robot to create a distributed system, aligning it with real-world scenarios. Following that, we implement the MRCA algorithm in the robot and subscribe to all other robots' position and velocity channels. MRCA algorithm processes all these measurements and calculates the target waypoint or velocity for the robot at the next time step. Then, it sends the new waypoint/velocity to PX4 as a command to be applied by the PX4 offboard component. Gazebo internally receives this actuator command and applies it to the robot. We demonstrate the architecture of our high-fidelity simulation setup on our project webpage.