

# Method of UAV Swarm Control Using a Radioelectronic Rangefinder with Formal Safety Verification

Anatoliy Doroshenko<sup>1,2,†</sup>, Oleksii Beketov<sup>1,\*</sup> and Ruslan Shevchenko<sup>1,†</sup>

<sup>1</sup>*Institute of Software Systems NAS of Ukraine, Glushkov 40, 03187 Kyiv, Ukraine*

<sup>2</sup>*National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Beresteiskyi 37, 03056 Kyiv, Ukraine*

## Abstract

Under multitasking conditions, the efficiency of unmanned aerial vehicle (UAV) operators depends on the ratio of UAVs coordinated simultaneously. The objective of this work is to determine the minimum set of technical tools required to manage a group of UAVs. A control method is proposed that determines the distance to the group's leader using a time-of-flight (ToF) radio ranging engine, followed by iterative calculation of relative coordinates by the group members without reliance on a global geopositioning system. This method enables participants to control their distance from the central UAV, thereby implementing a collective leader-follower swarm model. A basic coordination algorithm for a two-member swarm was developed, and a simulation of joint movement was conducted using the Webots robotic systems simulation environment. The advantages of the proposed approach include swarm scalability, the ability to disperse and regroup by varying following distances, independence from global coordinate systems, and the simplicity of system control organization.

The algorithm's basic properties are formalized in the Anthill specification language as an executable specification. Anthill is a many-sorted algebraic specification system devoted to gradual formalization: we can formalize parts or aspects of a system rather than the whole, attach proof constructs to rules (with discharge by an external SMT solver, by built-in SLD resolution, or as a structured proof), and generate code from the specification in the project's programming language. In this case, we prove the safety property — the inter-vehicle distance stays within a bounded interval for the simulated leader trajectory — under two stated assumptions: a triangle inequality on the Euclidean norm and a kinematic envelope on the inner stabilization loop.

## Keywords

unmanned aerial vehicle swarm, UAV formation control, leader-follower model, formal verification, Anthill, Webots

## 1. Introduction

Currently, the problem of controlling unmanned aerial vehicles (UAVs) to minimize human operator involvement is highly relevant. To address complex missions, UAVs are deployed in groups, making it impractical to assign individual operators to each aircraft. To enhance operational efficiency, technologies are used to enable the operator to control a single leader UAV, while the remaining group members operate as followers in autonomous mode [1]. The operator may switch control between UAVs sequentially, for instance, as individual units are attrited or removed from the mission.

One approach to swarm formation is a model in which the leader remains within the direct line of sight of the other swarm members, who track its movements and follow it using surveillance cameras [2], lasers [3], or lidars [4]. The disadvantages of this approach include the requirement for tight swarm cohesion to ensure mutual direct visibility among participants, as well as complications arising under limited visibility conditions, such as nighttime or adverse weather.

Examples of various swarm movement topologies include the chain, V-formation, grid, and honeycomb. In a chain topology, where each preceding unit acts as a leader for the subsequent one, losing

*2nd Workshop "Intelligent information technologies" UkrProg-IIT'2026 co-located with 16th International Scientific and Practical Programming Conference UkrPROG'2026, May 5-6, 2026, Kyiv, Ukraine*

\*Corresponding author.

<sup>†</sup>These authors contributed equally.

✉ doroshenkoanatoliy2@gmail.com (A. Doroshenko); oleksii.beketov@gmail.com (O. Beketov);

ruslan.s.shevchenko@gmail.com (R. Shevchenko)

🌐 <https://rss.github.io/> (R. Shevchenko)

🆔 0000-0002-8435-1451 (A. Doroshenko); 0000-0003-4715-5053 (O. Beketov); 0000-0002-1554-2019 (R. Shevchenko)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

sight of the leader immediately results in a broken chain, loss of connectivity, and loss of controllability. Similarly, the loss of the primary swarm leader from view causes the remaining members to become disoriented. Restoring swarm connectivity in such cases is extremely difficult, particularly when following non-linear trajectories involving complex maneuvering. More sophisticated topological movement models require additional mechanisms to ensure collision avoidance between neighboring participants.

Another approach involves establishing a radio network among swarm participants [5] and utilizing it to exchange data, such as current coordinates, velocity, heading, and acceleration, as well as to transmit control commands. Data transmission over the network enables the implementation of complex swarm behavior models. The completeness of information about the state of the entire system — where each swarm participant is aware of the status of all other members — enables the decentralization of swarm control. The vulnerability of such systems lies in their susceptibility to jamming and spoofing.

Another control method involves UAVs moving along a pre-synchronized trajectory, guided primarily by GPS sensors and accelerometers [6]. In such a system, the UAVs do not communicate with each other and are essentially independent. This system does not constitute a coordinated swarm but is rather a cluster of UAVs. This method is widely used to organize drone shows.

## 2. Core of the Method

Suppose a swarm consists of  $N$  participants equipped with radio transceivers. The distance between swarm members is periodically calculated based on the time-of-flight (ToF) principle. By maintaining the capability to calculate these mutual distances, the participants adjust their trajectories to maintain predefined distances, thereby achieving coordinated swarm movement. Under this control scheme, the participants' absolute coordinates are not the determining factor; only the relative displacements with respect to the leader and other members are significant.

Distance measurement and communication can be performed using a specialized transceiver, such as the Semtech SX1280. Introduced in 2017, the SX1280 is based on LoRa long-range radio technology for low-power devices. Its key feature is an integrated ToF ranging engine, which enables line-of-sight distance measurement up to 1.7 km with a 0.6% margin of error [7]. To determine the distance between two transceivers, one transmits a signal to the second, which, upon reception, responds with an echo signal. The first transceiver measures the time interval until the response arrives and calculates the distance to the second transceiver, accounting for the speed of radio-wave propagation, the round-trip time, and the latency of the second module. Since the ranging module is hardware-based, the processing time — following proper calibration — is considered constant. This scheme avoids the need for clock synchronization between devices, which would be required for one-way non-echo communication, and mitigates the effects of Doppler shift.

In a swarm of two participants, the follower periodically measures the distance to the leader and adjusts its heading to maintain it within established limits. In this scenario, neither the relative nor the absolute coordinates of the leader are known to the follower.

For a swarm with a larger number of participants, the current swarm configuration can be determined using the trilateration method. Determining the relative coordinates of the participants — and thus the current swarm configuration — is carried out in three stages: in the first stage, the swarm members determine the mutual distances between one another; in the second, the acquired data is exchanged among the participants; and in the third stage, having a sufficient dataset, the participants calculate the current configuration of the swarm.

In the first stage, a full measurement cycle can be performed either with or without repetitions. In the case of repeated measurements, all participants calculate distances to one another; specifically, participant  $i$  measures the distance to  $j$ , and conversely,  $j$  measures the distance to  $i$ , thereby duplicating the measurements. The resulting excess data can subsequently be utilized for error correction and to enhance the accuracy of the configuration calculations. In measurements without repetitions, participant  $i$  might not calculate the distance to each of the remaining  $N - 1$  participants, but rather for a subset, such as  $\{j \mid i < j \leq N\}$ . This preserves the system's completeness, ensuring that the distance between

each pair of participants is measured exactly once.

After the data exchange, the participants receive aggregated information regarding the pairwise distances between all swarm members. Based on these distance measurements, the probable locations of the leader and other participants are calculated iteratively. By adjusting their velocity and heading, the participants maneuver to maintain a specified distance from the leader and preserve their relative positions. To ensure process efficiency, participants can be divided into clusters to perform distance calculations and exchange data in parallel [8]. This is particularly relevant for maintaining system responsiveness as the number of agents increases.

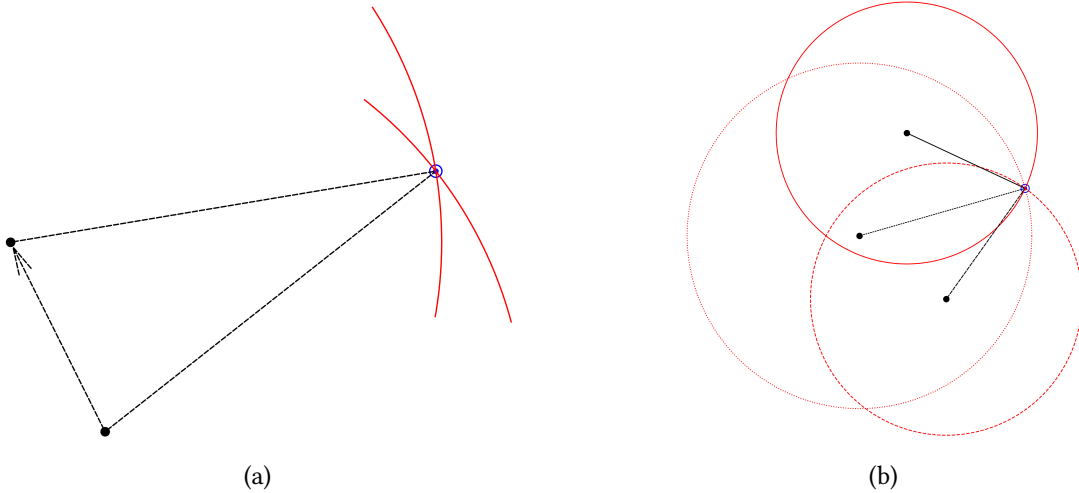
Next, consider different positioning scenarios. For simplicity, we will treat this as a planar problem, assuming that the participants are located in the same plane at a constant altitude. The velocity of the swarm participants will be considered negligible compared to the speed of radio wave propagation.

## 2.1. Stationary Beacon Localization

Consider the simplest case of stationary beacon localization. The problem involves the bearing of a fixed beacon, which periodically transmits a radio signal, by a moving observer as shown in Fig. 1(a). While in motion, the observer measures the distance to the beacon from various positions. The distance data implies that the beacon is located at a certain point on a circle of the corresponding radius, centered at the point from which the measurement was obtained:

$$(x_B - x_i)^2 + (y_B - y_i)^2 = r_i^2, \quad (1)$$

where  $(x_i, y_i)$  are the coordinates of the moving observer at step  $i$ ,  $r_i$  is the distance between the beacon and the observer at step  $i$ , and  $(x_B, y_B)$  are the beacon coordinates to be determined.



**Figure 1:** Stationary and moving beacon

After performing two distance measurements, the observer can determine the beacon's position as the intersection point of two circles. By formulating a system of two equations for the current and previous steps, a quadratic equation is obtained to determine the beacon's position, which, in the general case, yields two solutions. To uniquely determine the beacon's coordinates, the observer must perform a third measurement by assuming one of the obtained solutions and moving in a manner that either closes the distance to the assumed beacon location or increases the distance from it. Depending on the result, the assumption is either confirmed or refuted, in which case the second solution is accepted. Thus, the problem is uniquely resolved through three measurements. It is also possible for the solutions to coincide, indicating that the observer moved along the axis between their initial position and the beacon. There are no other possible variants for the considered case.

## 2.2. Moving Beacon Localization

Consider the inverse of the previous case, where the beacon is in motion while the observer tracks its trajectory as shown in Fig. 1(b).

A single fixed observer cannot determine the beacon's position because concentric circles centered at the observation point either do not intersect or, in the case of circles with identical radii, overlap at all points simultaneously. Therefore, we introduce an additional auxiliary observer. Two fixed observers define two circles that either intersect at two points or are tangent to each other if the beacon lies on the same line as the observers. To uniquely determine the beacon's position, a third observation point must be added, provided it is not collinear with the other two. Consequently, for a moving beacon, three stationary observers are required to uniquely determine its position, and its current location can be established with a single measurement cycle.

## 2.3. Cooperative Movement

The participant index  $i$  is used to compute the current spatial configuration from the coordinates of a trio of participants  $j$ ,  $k$ , and  $l$ . This trio  $\{j, k, l\}$  may be defined as common for all swarm members or selected individually, for example, from among the nearest neighbors.

Participant  $i$  sets their own coordinates as the origin of a local coordinate system  $(0, 0)$  and sets the coordinates of  $j$  as  $(r_{ij}, 0)$  or  $(0, r_{ij})$ . The coordinates of  $l$  are determined as the intersection of two circles: one centered at  $(0, 0)$  with radius  $r_{il}$ , and the second centered at the coordinates of participant  $j$  with radius  $r_{jl}$ . For determinacy, one of the two possible solutions is chosen, as the second solution is symmetric to the first and is therefore equivalent within a relative coordinate system. The coordinates of the remaining participants are calculated via trilateration relative to the coordinates of participants  $i$ ,  $j$ , and  $k$ . It should be noted that solutions may not exist due to measurement errors.

Participant  $i$  determines their motion vector by comparing and analyzing the relative displacements between the current and previous configurations.

## 2.4. Navigation Algorithm

Consider a movement direction selection algorithm for a follower in a swarm consisting of a leader-follower pair. Let  $\Delta_i = r_i - r_{i-1}$  be the difference in distance between the leader and the follower at the current and previous steps. The sign of  $\Delta_i$  indicates the nature of the distance change — whether the units are receding or approaching — while the behavior of the sequence  $\{\Delta_i\}$  indicates the rate of this change (accelerated, decelerated, or uniform). A decreasing sequence  $\{\Delta_i\}$  with negative values signifies accelerated convergence. The follower moves forward while simultaneously performing a turning maneuver, choosing the direction such that the sequence  $\{\Delta_i\}$  is non-increasing. According to this algorithm, if the heading is lost, the sign of  $\Delta_i$  becomes positive and the sequence  $\{\Delta_i\}$  increases, triggering the follower to initiate a turn. At a certain point in time, the follower's heading will be oriented toward the leader; assuming the follower's velocity is higher than that of the leader, the distance will begin to decrease, and the follower will start to close in on the leader.

## 3. Algorithm Simulation

The listing below presents the movement control algorithm for the follower:

```
1   yaw = Y;
2   while (1) {
3       r1 = r0;
4       r0 = get_distance();
5
6       d1 = d0;
7       d0 = r0 - r1;
8   }
```

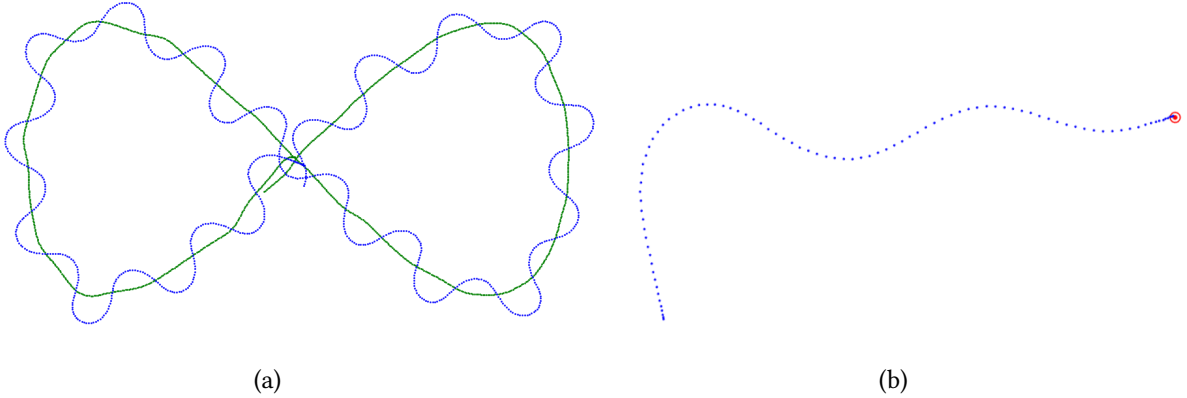
```

9      if (d0 < d1)
10         if (c <= 0) c--;
11         else      c = 0;
12     else
13         if (c >= 0) c++;
14         else      c = 0;
15
16     if (c < -T)    s = -1;
17     else if (c > T) s = +1;
18     else          s = 0;
19
20     if (c == 0 && s == -1)
21         yaw = -yaw;
22
23     Sleep(time_step);
24 }

```

In the provided code,  $r_0$  and  $r_1$  represent the distance at the current and previous steps, respectively;  $d_0$  and  $d_1$  are the current and previous members of the sequence  $\{\Delta_i\}$ ; the variable  $s$  determines the nature of the sequence  $\{\Delta_i\}$ ;  $T$  is a constant defining the threshold for registering a change in sequence behavior; and  $Y$  is a constant specifying the yaw. The described algorithm was modeled using the Webots [9] open-source development environment for emulating robotic systems.

The Webots environment provides primitives for modeling radio communication devices but does not account for the speed of radio wave propagation; consequently, it lacks the capability to model signal propagation delay. Due to this limitation, the data structure transmitted to determine distance contained the spatial coordinates of the leader. Subsequently, the distance was calculated based on the received coordinates rather than the signal reception time. This approach allowed for abstraction from physical environment modeling without affecting the subsequent calculations.



**Figure 2:** Moving and stationary target

Figure 2(a) illustrates the movement trajectories of a leader-follower pair, where the leader moves along a path defined by a closed curve of the form  $x(t) = \cos(t)$ ,  $y(t) = \frac{\sin(2t)}{2}$ , for  $\frac{\pi}{2} \leq t \leq 2\pi + \frac{\pi}{2}$ . At the initial moment, the participants are positioned at the center, facing southwest.

Figure 2(b) illustrates the trajectory of the follower in a scenario where the leader is stationary.

## 4. Partial Formalization in Anthill

The basic safety properties of the coordination algorithm described above — absence of collision and a bounded inter-vehicle distance — has been formalized in the Anthill specification language [10] and

discharged by an SMT solver. The same Anthill source is also the source of the C++ controller used in the Webots simulation.

Anthill is a many-sorted algebraic specification language. Every entity in the domain is described as a sort together with the operations that act on it. An operation can be implemented in an expression sublanguage, from which a code generator emits executable code in a target host language, C++ in our case.

Properties are stated as rules, which can have an attached proof construct. A proof can call an external SMT solver such as Z3 [11], use the built-in SLD resolution, or be given directly as a structured proof that composes named sub-rules. Each discharge produces a content-addressed witness. The kernel re-checks every witness on `anthill check`, recursing through composed witnesses and re-resolving cited rules.

#### 4.1. Formalization structure

The Anthill description of the algorithm is organized in three layers.

- mirror of the Webots C++ API: each Webots class used by the controller (GPS, Emitter, etc) is reflected as an Anthill sort with the same operations, so that the host-language API visible to the generated controller is the genuine Webots one.
- controller specification proper — the leader’s waypoint logic and the follower’s transponder-driven hill-climbing controller.
- the safety description: a base vocabulary (drone identity, distance bounds, kinematic assumptions) plus the transponder-specific obligations and their proofs.

The inner stabilization loop (the fixed-gain PID and motor mixing of the Mavic2Pro) is left outside this stack: it is carried verbatim from the reference implementation as a hand-written C++ file and treated as an assumption by the safety description.

The Webots interface layer was generated by a Claude Opus agent following the rules: each Webots class becomes a sort, each public method becomes an operation, and alongside each sort the agent emits an `Implementation` fact that binds the sort to the host C++ class. The codegen reads those facts and inverts the translation when emitting C++.

Each controller exports two operations: an effectful `step()` that performs the sensor read and motor write, and a pure `compute_controls()` that maps the current state to the desired `Controls`. The C++ controller used in the simulation is emitted from this layer. As a result of the compilation, we can receive a self-contained Webots project.

The safety description was authored separately from the controller. It introduces a base vocabulary — drone identity, distance bounds, kinematic assumptions, and abstract real-pose / observed-pose predicates — together with the two trusted axioms used by the proof (the triangle inequality on the Euclidean 3-norm and the kinematic envelope on the inner stabilization loop) and the transponder-specific obligations.

#### 4.2. Safety property

The safety contract states that the inter-vehicle distance must lie within a bounded interval  $[d_{\min}, d_{\max}]$ . The lower bound expresses the absence of collision; the upper bound expresses formation cohesion (loss of radio range above  $d_{\max}$ ). For the transponder controller, these two halves of the contract have a different mathematical character.

Absence of collision is a per-step inductive invariant: once the follower enters its capture radius, it cuts pitch and yaw to zero, so the only remaining motion is the leader closing the gap, bounded per tick by the product of leader velocity and tick duration.

Formation cohesion is *not* a strict per-step invariant: the hill-climbing yaw-flip search drifts the follower outward transiently while it locates the gradient. The accurate formulation is a bounded excursion: from any post-armed state the controller forces a yaw flip within a bounded number of

consecutive bad ticks, and the trajectory stays within a bounded excursion above  $d_{\max}$  during that window. The bound itself is derived in the proof.

### 4.3. Proof and assumptions

The proof rests on two trusted axioms; everything else is mechanically discharged. The first is the triangle inequality on the Euclidean 3-norm — geometry of  $\mathbb{R}^3$ , not derivable from anything Anthill knows about Float arithmetic. The second is a kinematic envelope on the inner stabilization loop: the load-bearing assumption that the carried-verbatim Mavic2Pro PID keeps each vehicle's velocity below a stated bound.

The proof first defines a per-tick upper bound  $\delta_t$  on the change of the real inter-vehicle distance. A structured proof decomposes  $\delta_t$  into two contributions and closes each by Z3 in linear real arithmetic (LRA):  $(v_L^{\max} + v_F^{\max}) \cdot T_c$  from physical motion, and  $2 \epsilon_t$  from the scalar ranging noise.

With  $\delta_t$  established, the absence of collision is simply half. It splits into a base case at the launch geometry and a per-step inductive step.

The base case is stated as a top-level public claim that denies any reachable real configuration with  $d < d_{\min}$ :

```

1 rule safety_no_collision_transponder(?w)
2   :- reachable_real(?leader_real, ?follower_real),
3     position_distance(?d, ?leader_real, ?follower_real),
4     DistanceBounds(d_min: ?d_min, d_max: ?),
5     ?d < ?d_min,
6     ?w = ?d
7
8 proof safety_no_collision_transponder
9   by z3(logic: "QF_NRA")
10 end

```

The discharge uses quantifier-free nonlinear real arithmetic (QF\_NRA) because position\_distance is squared; The inductive step is the per-step obligation lower\_violation\_transponder: it succeeds whenever some  $(d_{\text{prev}}, \text{step})$  inside the strong-invariant interval  $[d_{\min} + \delta_t, d_{\max} - \delta_t]$  drives the next distance below  $d_{\min}$ , and the proof asks Z3 to show this rule has no solution:

```

1 rule lower_violation_transponder(?d_next)
2   :- DistanceBounds(d_min: ?d_min, d_max: ?),
3     transponder_step_distance_bound(?delta_t),
4     ?d_low_bound = ?d_min + ?delta_t,
5     ?d_low_bound <= ?d_prev,
6     abs(?step) <= ?delta_t,
7     ?d_next = ?d_prev + ?step,
8     ?d_next < ?d_min
9
10 proof lower_violation_transponder
11   by z3(logic: "LRA")
12 end

```

By induction on tick, the two pieces lift the launch geometry refutation to every reachable  $k$ .

The induction step for the upper bound is the non-trivial half. It is encoded as a per-step refutation parametric in a window width  $N\delta_t$ : starting from the shrunk headroom band  $[d_{\min} + N\delta_t, d_{\max} - N\delta_t]$  with steps of magnitude at most  $N\delta_t$ , the next distance must not exceed  $d_{\max}$ :

```

1 rule bounded_excursion_upper_violation_transponder(?d_next)
2   :- DistanceBounds(d_min: ?d_min, d_max: ?d_max),
3     excursion_distance_bound(?excursion),
4     ?d_low_bound = ?d_min + ?excursion,
5     ?d_high_bound = ?d_max - ?excursion,
6     ?d_low_bound <= ?d_prev,

```

```

7      ?d_prev <= ?d_high_bound,
8      abs(?step) <= ?excursion,
9      ?d_next = ?d_prev + ?step,
10     ?d_next > ?d_max
11
12 proof bounded_excursion_upper_violation_transponder
13   by z3(logic: "LRA")
14 end

```

The strict per-tick claim  $d \leq d_{\max}$  does not hold for this controller, which is why the obligation is stated in this bounded-excursion form: the hill-climbing yaw-flip search drifts the follower outward transiently while it locates the gradient. Stating the obligation in this form exchanges one problem for another: we now have to justify the constant  $N$ . Picking  $N$  by hand would be a guess.

To justify  $N$  we descend to the controller's discrete state. The follower's state is a record `TransponderFollowerState` carrying, among other fields, an integer counter *upc*; the operation `step()` takes the current state and a sensor reading and returns the next state. The relevant fragment of `step()` updates *upc* as follows:

```

1 let new_upc =
2   match state.prev_delta
3   case none -> 0
4   case some(pd) ->
5     if new_delta < pd
6       then if state.upc <= 0 then state.upc - 1 else 0
7       else if state.upc >= 0 then state.upc + 1 else 0

```

The follower spec uses  $upc \in [-6, 0]$  as the *post-armed* regime — the window between detecting sustained improvement ( $upc \leq -6$  raises a regime flag) and the next yaw flip (*upc* returning to 0 from below triggers one). On a bad-step transition (the `else` branch above), the safety description abstracts the update to  $upc_{next} = upc + 1$  — the worst-case increment, which the actual code respects. A bound on  $|upc|$  within that regime is therefore a bound on the number of consecutive bad ticks before the next flip, and hence on  $N$ .

This is what the ranking meta-tactic establishes. With the well-founded measure  $R = -upc$ , the tactic dispatches two LIA sub-queries on closed integer formulas over *upc* alone:  $R$  is non-negative on every post-armed state (`bound_violation_transponder` has no solution —  $upc \in [-6, 0]$  contradicts  $upc > 0$ ), and  $R$  strictly decreases on every bad-step transition (`decrease_violation_transponder` has no solution —  $upc_{next} = upc + 1$  contradicts  $upc_{next} \leq upc$ ). The tactic then composes the two witnesses into one for the trivial parent rule.

```

1 rule bound_violation_transponder(?w)
2   :- ?upc >= -6, ?upc <= 0,
3     ?upc > 0,
4     ?w = ?upc
5
6 rule decrease_violation_transponder(?w)
7   :- ?upc >= -6, ?upc < 0,
8     ?upc_next = ?upc + 1,
9     ?upc_next <= ?upc,
10    ?w = ?upc
11
12 rule post_armed_excursion_bound(?w) :- ?w = 1
13
14 proof post_armed_excursion_bound
15   by z3(tactic: ranking(boundedness: bound_violation_transponder,
16                        decrease: decrease_violation_transponder),
17        logic: "LIA")
18 end

```

Both sub-queries come back unsat, so the controller cannot remain in a post-armed bad-step regime for more than 6 consecutive ticks before flipping heading. The bound is exposed as a fact, where the literal 6 is not a guess but a witness emitted by the ranking proof. The bound  $N = 6$  is trajectory-specific: it holds because, on the simulated leader trajectory, the post-armed window is entered at depths within  $[-6, 0]$ . A different leader trajectory may force the ranking proof to be re-discharged against a different upper bound on  $|upc|$ . Plugging  $N = 6$  into the parametric per-step refutation above closes the bounded excursion safety claim: the real distance can exceed  $d_{\max}$  only transiently during a yaw-flip search, by at most  $6 \delta_t$ , before the controller drives the trajectory back inside.

The full proof, including the trusted axioms, the structured step-distance lemma, the per-step refutations, and the ranking discharge, ships with the Anthill distribution in the `examples/webots-modelling/` directory under the leader-follower example.

#### 4.4. Anthill specifications during project development

Code and proof share one knowledge base. If an engineer rewrites the body of the leader's `compute_controls`, the operational semantics change, the derived  $\delta_t$  is recomputed, every Z3 sub-query under the bounded-excursion ranking proof is re-run, and the LRA discharge of the no-collision obligation re-runs. Proofs whose dependencies have shifted are marked stale by the dependency graph and re-checked on the next `anthill prove`; the C++ code generator emits a new controller from the same edit. The two artifacts: proof and source code, are always synchronized as projections of the same specification.

### 5. Conclusions

A self-coordination approach for a UAV swarm in open terrain is proposed, based on an inter-agent distance control mechanism that does not require global geopositioning. A fundamental interaction algorithm for a two-member swarm was developed, and its performance was validated through simulation.

Future work will involve the development and simulation of an expanded algorithm, which will have trajectory-independent bounds: the reformulation under exploration replaces the load-bearing speed-envelope axiom with a per-tick acceleration tracking axiom for the inner loop, and introduces a separate configured acceleration limits for the leader and follower. The strict-advantage condition required is that the follower's acceleration limit exceed the leader's by a margin that dominates the inner-loop's tracking error. Under that formulation,  $N$  becomes a function of the configured limits alone, and the bounded-excursion claim becomes universal over leader trajectories respecting the configured budgets.

The same direction also opens the door to richer follower controllers — for instance, a proportional controller on the scalar transponder reading that exploits the strict-advantage budget on demand. We plan to extend the algorithm to an arbitrary number of participants in both planar and three-dimensional space.

### Declaration on Generative AI

A Claude Code was used to verify the extracted proof fragments and the correctness of the explanation. Grammarly was used for language grammar checking. After using these tools, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

### References

- [1] H. T. Do, H. T. Hua, M. T. Nguyen, C. V. Nguyen, H. TT. Nguyen, H. T. Nguyen, N. TT. Nguyen, Formation control algorithms for multiple-uavs: A comprehensive survey, EAI Endorsed Transac-

- tions on Industrial Networks and Intelligent Systems 8 (2021) e3. URL: <https://publications.eai.eu/index.php/inis/article/view/223>. doi:10.4108/eai.10-6-2021.170230.
- [2] M. Bastourous, J. Al-Tuwayyij, F. Guérin, F. Guinand, Image based visual servoing for multi aerial robots formation, in: 2020 28th Mediterranean Conference on Control and Automation (MED), 2020, pp. 115–120. doi:10.1109/MED48518.2020.9182942.
  - [3] M. Rui, Z. Zhiyan, Z. Lyu, X. Luo, L. Zi, S. Cancan, Y. Zang, W. Liu, R. Jiang, Laser tracking leader-follower automatic cooperative navigation system for uavs, International Journal of Agricultural and Biological Engineering 15 (2022) 165–176. doi:10.25165/j.ijabe.20221502.6350.
  - [4] U. Seidaliyeva, L. Ilipbayeva, D. Utebayeva, N. Smailov, E. T. Matson, Lidar technology for uav detection: From fundamentals and operational principles to advanced detection and classification techniques, Preprints (2024). doi:10.20944/preprints202410.1344.v1.
  - [5] S. Pandi, F. Gabriel, O. Zhdanenko, S. Wunderlich, F. H. P. Fitzek, Meshmerize: An interactive demo of resilient mesh networks in drones, in: 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), 2019, pp. 1–2. doi:10.1109/CCNC.2019.8651795.
  - [6] Verge Aero, How drone shows work: An overview, 2024. URL: <https://docs.verge.aero/drone-show-technology/how-drone-shows-work-an-overview/>, accessed: 2026-05-04.
  - [7] F. Rander Andersen, K. Dilip Ballal, M. Nordal Petersen, S. Ruepp, Ranging capabilities of lora 2.4 ghz, in: 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), 2020, pp. 1–5. doi:10.1109/WF-IoT48130.2020.9221049.
  - [8] P. I. Andon, A. Y. Doroshenko, V. G. Akulovskiy, P. A. Ivanenko, O. A. Yatsenko, Formal and adaptive methods of constructing high-performance parallel programs, Kyiv, Naukova Dumka, 2023. doi:10.15407/978-966-00-1809-9, in Ukrainian.
  - [9] O. Michel, Webots: Professional mobile robot simulation, Journal of Advanced Robotics Systems 1 (2004) 39–42. URL: <http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf>.
  - [10] R. S. Shevchenko, A. Y. Doroshenko, O. A. Yatsenko, Anthill: a progressive formalization language for agent-supported software projects, Problems in Programming (2026) 4–13. URL: <https://pp.isoftware.kiev.ua/index.php/ojs1/article/view/887>.
  - [11] L. de Moura, N. Bjørner, Z3: An efficient SMT solver, in: Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), volume 4963 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 337–340. doi:10.1007/978-3-540-78800-3\_24.