Information, Computing and Intelligent Systems, 2025, No. 6, 152 – 163

UDC 004.67, 004.38 https://doi.org/10.20535/2786-8729.6.2025.333220

DRONE SWARM CONTROL MODEL BASED ON HIGH-LEVEL PETRI NETS

Valentyn Ivankov *
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine
https://orcid.org/0009-0001-9835-8486

Mykhailo Novotarskyi
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine
http://orcid.org/0000-0002-5653-8518

*Corresponding author: valentyn.ivankov@gmail.com

The rapid growth of unmanned aerial vehicle (UAV) applications in the modern world imposes significant demands on the reliability of control logic. An error in the sequence of stages can lead at best to inefficient battery usage or violations of airspace regulations, and at worst to an accident with loss of the vehicle and potential harm. Control is usually implemented using scripts or behavior trees, which complicates maintenance. The reason is that the size of the source files quickly increases, and when it becomes necessary to add new functionality or modify existing logic, there is a risk of introducing vulnerabilities by failing to account for all possible situations. This is why High-Level Petri Nets (HLPN) were chosen, as this method addresses the problem of formally describing the control system and allows the system to be easily scaled or modified in any way.

The aim of the study is to develop and validate a model based on HLPN that will serve as the single source of truth for UAV swarm control. In the proposed model, the places correspond to flight stages, and the tokens carry numerical parameters such as battery charge, coordinates, and telemetry. Thus, a single scheme simultaneously describes discrete events and constraints. For each transition, conditions are formalized to verify the possibility of its execution, such as checking the minimum required battery level or verifying location.

The methodology includes several stages. First, the network structure is formally defined. Then, based on this structure, a Python model is built that implements the developed network, controls movement between states, and ensures the correct sequence of transition firings. After developing the model, testing and analysis of the obtained results are performed.

The results show that using HLPNs to build a model for verifying commands in a discrete mode indeed ensures a correct description of transitions between states and increases the reliability and survivability of the developed control system model, while also significantly reducing maintenance efforts. The developed model is easily adaptable to route changes, addition of sensors, or functional expansion.

Key words: Unmanned Aerial Vehicle, Drone, High-level Petri Net, simulation, control.

1. Introduction

This article is about information systems, autonomous control systems, and formal methods. It aims to solve the problem of describing and verifying the logic that drives an Unmanned Aerial Vehicle (UAV) swarm, ensuring that every step of a mission is both correct and easily auditable. Autonomous UAVs are part of the broader field of intelligent systems. They are the machines whose software decisions have direct physical effects. Inside this area, the work focuses on the formal modeling of discrete-event behavior. Formal models are great because they allow us to test ideas in mathematics first, then in simulation, and finally on hardware.

For instance, delivery UAVs must switch between clearly defined flight stages. If the controller skips, repeats, or mixes steps, the craft can crash, violate air-space rules, or waste energy. Traditional

code bases grow into legacy, hard-to-debug systems, and popular behavior-tree editors cannot provide strong guarantees. The general scientific problem is, therefore, the lack of a simple, verifiable model for mission-level control that scales from whiteboard sketches to executable code.

We argue that HLPNs are well-suited for this role. In a High-level Petri Net (HLPN), circles (*places*) mark flight stages; bars (*transitions*) mark commands; a dot (*token*) shows the current state [1]. Tokens in HLPNs can carry numbers, such as battery level and GPS coordinates, allowing the same picture to capture both discrete events and continuous limits [2]. Simulations can execute the net, explore every possible path, and prove constraints such as the UAV always returning to the base or never lifting off with insufficient energy. Safe, explainable control of small UAV swarms is a live challenge for spheres such as agriculture, inspection, and emergency for the analysis, but logistics. Regulators are increasingly requiring evidence that autonomous aircraft will follow prescribed steps and stay within established security margins. Formal models, such as HLPNs, provide that evidence at a modest cost [3].

Therefore, the presented research is important, because it fulfills an urgent industrial need, and in a meanwhile participate in an academic discussion regarding usage of the graph-based formal methods for robotics.

2. Literature review and problem statement

Modern UAV operations still use the canonical stage chain *Ready* → *Take-off* → *Route* → *Land*, and any skipped, repeated, or confused stage can lead to crashes, air-space violations, battery waste, or mission aborts. In [3] authors highlight the same issue but for Autonomous Underwater Vehicles. The only difference is the operation space, water vs air. Because of using the tree analysis approach, if a failure occurs, it is hard to understand whether the current mission can be successfully finished [3]. Also, it is worth noting that the control logic for UAV swarms is rarely centralized. In practice, flight-control logic is often a mix of handwritten state machines and ad-hoc scripts, fragmented across the entire code base with various validation rules.

Additionally, essential parameters such as battery, payload, route geometry, and wind drift change from mission to mission. Moreover, as noted in [4], stochastic effects, such as failures, must also be included in the model, which adds a new problem. In conclusion, all of the above require extra code checks that pile up and quickly become unmanageable. The challenge escalates once a swarm flies beyond the visual line of sight. After this, every in-air command must be provably valid. On the one hand, behavior-tree-based scripts can offer quick prototyping but are essentially unable to block all illegal commands during flight, leaving a safety gap between simulation and reality and introducing a "third-party-risk gap" [5].

Some research shows that HLPNs can solve the problem above because they handle discrete flight modes and continuous numeric resources in a single graph. In [4], the authors use colored tokens to model inspection drones, including stochastic repair times, and demonstrate that CPN enables easy reconfiguration of inspection patterns. In [6], the idea is extended to wireless-power UAV fleets, modeling both energy flow and information latency in a Petri net. Furthermore, in [7], the authors introduce a spatial-temporal hybrid Petri net that controls physical motion for heterogeneous UAV clusters and formally verifies attack-mission timelines. These studies demonstrate that Petri nets can effectively express the rich behavior of UAVs; however, in all these cases, the nets primarily serve as analysis tools and lack essential safety features. Also, large-scale UAV swarms are dependable on communication links, resource reserves, and real-time fault isolation. As mentioned in [9], connection outages are instead an external surprise rather than an integral part of the mission [9]. Additionally, [6] demonstrates that wireless-power missions must consider energy utilization and information latency. The paper [10] indicates that we can detect intermittent sensor faults online; however, they usually do not automatically reroute or replan the mission.

The analysis of literary sources highlighted the existence of several unresolved problems that arise when controlling a swarm of drones.

The main problem is that for a swarm of drones, there is a need for strict formalization of the behavior of the swarm members since any missed or uncoordinated step can lead to accidents, airspace violations, loss of battery power, or mission interruption.

Another critical issue is ensuring adaptability and safety. There is a need to easily scale the control system from a few drones to hundreds. The drone control system must support all safety checks, despite changes in internal conditions (such as battery level or stochastic influences).

3. The aim and objectives of the study

The object of the research is the process of developing a HLPN based control systems for a swarm of UAVs operating in a dynamically changing environment.

The aim of this research is to implement a control model for the UAV swarm based on HLPN. This allows to introduce a strict formal description of behavior while ensuring scalability and ease of extension. Basically, we should be able to use the model not only for the analysis but also be able to use it for mission flow control.

To achieve this goal, the following tasks are set:

– To develop an HLPN based, scalable, safety-compliant control model that ensures strict control over UAV swarm members to prevent accidents, like battery depletion, or mission failure. Also, it should support seamless expansion from a few units to hundreds while maintaining adaptability to internal condition changes. Finally, it should be easily adaptable for route changes, addition of sensors or other functional expansion with as less maintenance efforts as possible.

– To perform modeling and then evaluate its effectiveness by concluding a comparative analysis of the modeling results.

4. Materials and Methods for developing a UAV control system using HLPN

The research methods of the article are the methods and models for adaptive control and safe reconfiguration of UAV swarms in a dynamically changing environment, particularly using HLPNs for formal behavior modeling, safety assurance, and swarm coordination under environmental uncertainties.

4.1 The object and hypothesis of the study

This study aims to develop a drone swarm control model based on HLPNs. The first step in building the model is to define an HLPN, which formally describes the behavior of a drone in a swarm as a bipartite graph, the nodes of which are represented by *places* and edges represented by *transitions*. Each *place* contains a defined data structure that specifies the current state of the drone, and the *transition* with *Guard* determines the conditions for modifying the state.

The second problem identified in the literature review is ensuring the adaptability and safety of the swarm in conditions of dynamic environmental change. Improving current results in this direction requires further research and application of artificial intelligence technologies. In this work, this task will be solved by aggregating a set of HLPNs of swarm participants into a single model at a new hierarchical level, thereby ensuring coordination of swarm participants by eliminating critical situations.

To do this, we use transition *Guards* for critical transitions. These rules encode safety checks, such as minimum energy for takeoff and geo boundaries, ensuring that only legal commands can be fired, even when noisy telemetry or incorrect operator input is present. We conduct a reachability analysis to demonstrate that the conditions defined in the previous paragraph are met. The pre-flight checks are conducted so that the UAV never leaves the *Ready* stage in an invalid or nearly invalid state (i.e., with an insufficient battery level to perform the mission). Every UAV in the swarm eventually reaches *OnBase*, and the battery charge during the mission always remains within bounds. Finally, we implement the model work simulation, which records each firing sequence, allows for fast what-if tests, and leaves the ability for later experiments. A simulation code traverses the network, logs each firing sequence, and leaves hooks for later hardware-in-the-loop tests. A mission serves as

the running example, and we compare the modeling effort, clarity, and verification cost. Thus, by placing resource tokens, stochastic failure transitions, and transition guard checks in one executable graph, Petri nets move modeling from manual audits into a live and adaptive safety mechanism that keeps the swarm in valid states all the time.

To summarize, the objectives described above support the primary goal of the research. They help ensure that the developed model will deliver a simple yet fault-tolerant control system for the UAV swarm that can be easily expanded with new functionality while maintaining provable safety.

4.2. Justification of the concept of controlling UAV swarms using Petri nets

We decided to choose HLPNs because they combine the clarity of classical Petri nets with typed tokens that can store battery charge, 3D position, and mission flags in a single model. It allows one diagram to express both mode switches (discrete *places*) and resource limits (*Token* attributes), satisfying the "single source of truth" goal set in paragraph 2. Petri nets natively support concurrency and blocking guards, which closely match the reality of UAVs that must arbitrate between navigation, safety, and payload tasks. Compared to conventional control logic, such as Finite-State Machines (FSMs) or behavior trees, HLPNs offer two technical advantages.

In an FSM or a behavior tree, the mission graph is implicit: states are scattered across functions, while event guards are located in many if-else clauses. Adding a new branch (for example, "return if low battery") involves tracing many code paths to ensure that no unreachable or cyclic states appear. A HLPN, in contrast, renders every flight stage as a place and every command as a transition, so the complete set of paths is visible in one diagram [5]. Formal reachability tools are built-in and can analyze that diagram to ensure constraints, such as "the UAV always reaches base" or "cannot take off with less than 75% battery". Consequently, design reviews focus on a single artifact rather than scattered source files.

In non-Petri frameworks, quantitative limits (such as battery level or coordinates) live in ad-hoc variables that must be checked repeatedly [6]. Petri nets allow the encapsulation of those numbers into the token itself [7]; guard functions operate on the live token, so every transition fires only when its pre-conditions are valid. This tight integration shortens debug cycles and allows researchers to improve the controller gradually (by adding timed or stochastic extensions, for example) without switching tools.

In summary, while FSMs and behavior trees are quick to prototype, HLPNs provide a mathematically grounded, single-source model that scales easily. It makes them the stronger choice for any project that needs to control a UAV swarm and requires different regulatory approvals and operation safety.

4.3. Description of the HLPN model of the control system and tools

A Petri net is formally defined as a tuple, $N=(P, T, A, G)$. *Places* (or *States*) $P=\{$*Ready, Takeoff, EnRoute, AtTarget, Returning, OnBase, Charging*$\}$ correspond to flight stages. *Transitions* are bridges between mission steps: $T=\{$*CanTakeOff, TakeOffToEnroute, IsEnRoute, IsAtTarget, IsMissionComplete, IsReturning, IsOnBase, ChargeI*$\}$. *Arcs A* connect *places* with *transitions*. *Guards G* are boolean functions that accept the token as a parameter and verify if the transition firing is allowed. The logic of these *Guards* is reflected in the name of *transition*. For example, for *IsAtTarget* the function compares drone coordinates with mission coordinates. Overall, this approach ensures that every step is completed, and the system goes only from a valid to a valid state. The changes in *Token* status will be handled automatically as well.

There are a few tools that can be a good choice for HLPN prototyping. We will consider the platform, programming language, functionality, and the possibility of integrating with other tools. For example, visual simulations are usually performed in the Gazebo GUI, while control commands are typically sent from ROS2 [8]. Both tools are available on Linux, with only limited support for Mac OS and Windows. Regarding UAVs, most software also offers better support for Linux, making the

platform particularly important in this context. Additionally, the modeling tool must support colored HLPN; otherwise, the logic will be incomplete.

In Table 1, we mainly chose scientific tools (because some tools are business-oriented). The graphic interface is not considered a very important factor because there are many third-party libraries that can help with it, such as NetworkX or GraphViz. Additionally, the license should permit the use of the tool freely, allow for modification if needed, and permit the publication of results obtained with it. The great examples of it are MIT or LGPL.

Table 1. Comparing Petri net tools

Tool	Platform	HLPN support	State-space analysis	Integration & API	License	Suitability
CPN IDE	Eclipse-based IDE	Basic hierarchy, timed extensions limited	Basic reachability; relies on external solvers	Java/Eclipse plug-in system;	GPL	Lightweight; integration in Java/Eclipse; Less mature analysis features
PIPE 5	Java desktop	Basic colours via GSPN	Partial (analysis modules in PIPE 4 only)	Java API, CLI; no Python bridge	MIT	Easy drawing, but HLPN and analysis still catching up; Python glue code needed
Snakes	Pure Python library	Coloured nets, Python objects as tokens	Plugins for reachability, model-checking	Directly embedded in the same codebase; seamless with NumPy, NetworkX, etc.	LGPL-3	lightweight, headless, programmable
TINA	GUI (nd) and C/C++ analyzers	Time Petri Nets (no full HLPN support)	State space, symbolic, temporal logic (CTL/LTL), structural analysis	CLI tools, PNML import/export; no Python bridge	Free	Efficient and rigorous for TPN; Less expressive than CPN Tools; Split utilities instead of single IDE
PetriObj Model Paint	Java desktop	Object-oriented and stochastic multichannel Petri nets)	Scalable discrete-event simulation; graphical editor, mean token counts, buffers, final state;	Java API; no Python bridge	MIT	Provides both visual editing and executable object-oriented models, however, is limited to desktop only

Because the surrounding simulation (e.g., ROS2 node) is already written in Python, Snakes eliminates language barriers. Therefore, the places, transitions, guards, and other elements like

monitoring or cloud-integration [9, 10] can be integrated directly into it. Additionally, its plugin system further enables future extensions, such as timed networks or stochastic firing weights, without requiring a switch to other tools. Finally, Snakes is known for its execution and verification backbone, which is suitable for the proposed HLPN-based UAV controller.

4.4. Development of a mathematical model and formal Petri net for swarm operations

To monitor mission health for a fleet of n UAVs, we propose the following model that aggregates three key parameters for every craft: battery reserve, route length, and malfunction probability. Power consumption is often considered a bottleneck because everything depends on it: moving and controlling channels. [6] The following formula explains the energy part of the model:

$$B = 100 - \frac{1}{n}\sum_{i=1}^{n} b_i, \tag{1}$$

where b_i is the current battery level (in %) of the *i*-th UAV; *B* thus measures the average energy deficit of the swarm. This value indicates that if it is high enough, the UAVs will not be able to return.

Another critical parameter is operation range, L , the following formula allows to calculate it.

$$L = \frac{1}{n}\sum_{i=1}^{n} l_i, \tag{2}$$

where l_i is the route length int i-th UAV, it measures the average path of the swarm, so the higher it is, the better is the result.

The probability of a UAV malfunction p_i is determined by the formula (3):

$$p_i = \beta\left(\frac{l_i}{l_{max}}\right)^2 (100 - b_i)^2, \tag{3}$$

where β is a relatively small value that keeps the value realistically small, so when the route is 80% of the max, and the battery is 80%, the probability of malfunction will be 0.2. The quadratic dependence on both distance fraction and energy deficit captures the intuition that faults are exponentially likelier when a heavily loaded craft starts a long leg with little reserve.

4.5. The model of UAV swarm control system based on HLPN

To control the work, we use a control system based on Petri nets. As a result, the developed prototype of the method is tested for reliability. According to the problem statement, the Petri net (Fig. 1.) controls several markers (in the case of UAVs). In the figure, the system is at stage 1.

The first prototype of the model contained only the four core transitions: *Start* → *Take-Off* → *Move* → *Land. Additionally, it* did not include an *auto-return* branch that could force the UAV to abandon its task and head back as soon as safety margins (such as battery and link quality) were violated. Simulation results showed that the minimalist net completed around 76% of flights in the 100–UAV test set. The remaining exhausted their batteries during the initial states or got into an undefined state during the mission.

Because the initial Petri net structure lacked a guarded transition, the reachability graph exposed dead-end markings. It was confirmed at design time that a security analysis is needed to check where most failures occur. This weakness motivated the revised model (Fig. 1) in which a *ProblemOccured* transition was introduced. It can be fired during *Takeoff* or *IsEnRoute*. This fact guarantees that, eventually, every movable state marking contains a path to *IsOnBase*. Subsequent experiments increased the success rate, validating the design change and illustrating how a single well-placed transition can enhance mission-level reliability.

Stages of the control system.

Stage 1. All tokens (3 UAVs in our case) are in *Ready* place. If the condition for the *TakingOff* transition is met, tokens move further and are put in *the Takeoff* place.

Stage 2. Depending on takeoff, two scenarios are possible:
– If everything is OK, *the TakeOffToEnRoute* transition is fired, and tokens are put in stage 3
– If *TakeOffToEnRoute* is not allowed, tokens are put in stage 5.

Stage 3. Depending on the UAV state, there are three possible cases:
– The target is not reached, and there are no problems; fire *EnRoute* and stay at this stage.
– The target is reached, fire *AtTarget,* and move to stage 4.
– If the target is not reached and problems occur, move to stage 5.

Stage 4. According to the plane mission, action is done, and tokens move to stage 5.

Stage 5. It is similar to stage 3 but with two possible cases:
– The base is not reached, and there are no problems; fire *Returning* and stays at this stage.
– The base is reached, fire *OnBase,* and move to stage 6.

Stage 6. Charging the UAVs

Stage 7. If the transition guard condition is met, fire *Charged* and move to stage 1.

There are 3 *States* with multiple possible *transitions,* however they are mutually exclusive. For example, if the problem was detected then it is impossible to switch from *TakeOff* to *IsEnRoute.* The transition *Guard* in *TakeOffToEnroute* will not allow it. Similarly, switches from *IsEnRoute* to *IsAtTarget* or from *IsReturning* to *IsOnBase* are possible only if there are no problems and target or base coordinates were reached. The transitions are multichannel, meaning all tokens will be moved if it is allowed. The system aggregates all UAVs as *Tokens* and manages them simultaneously.

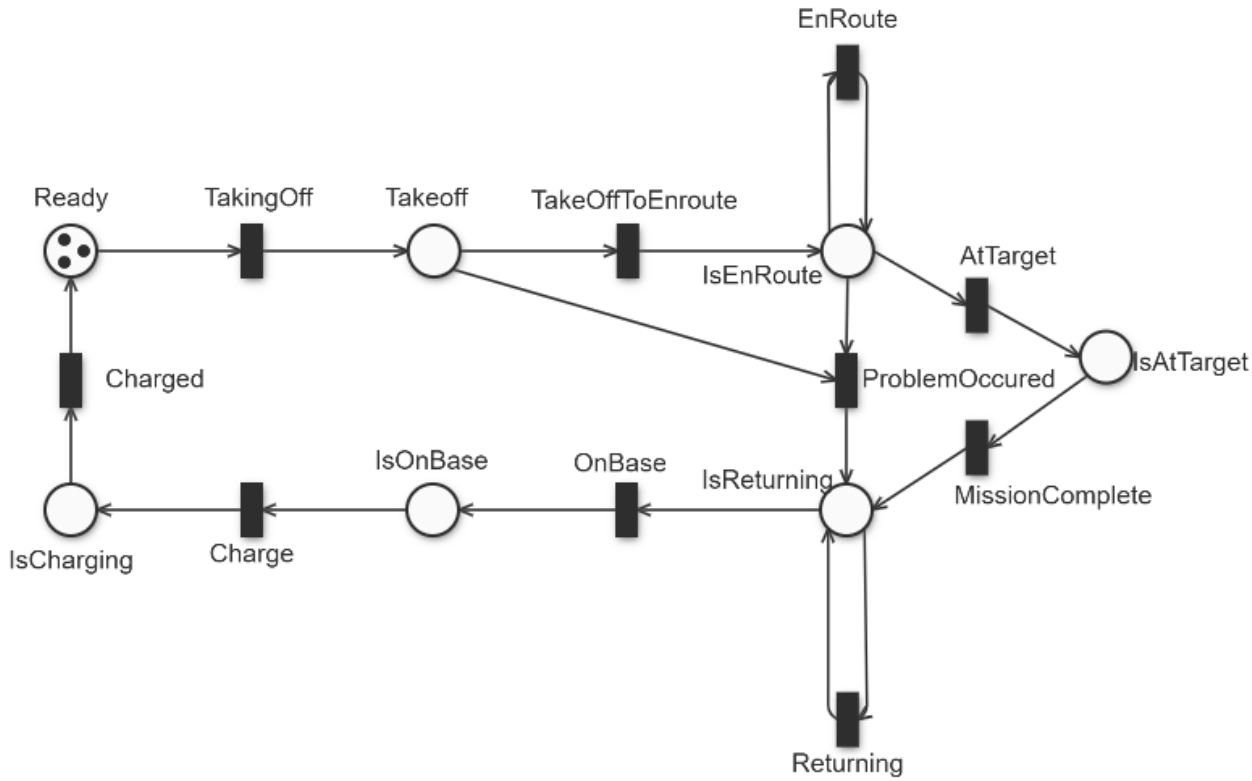

Fig. 1. Swarm operation control Petri net

Although the current control net contains only seven states, this is rather a strength than a limitation. A resulting HLPN is very modular: adding a new state and transition guard means attaching an extra function to an existing place while introducing a new maneuver, such as "*ChangeLeader*"

or "*ChangeFormation*" Additionally, the control net can operate in both edge and cloud modes, which increases reliability. The original net remains unchanged, and its previously verified properties still hold. Suppose the controller expands beyond a single scenario, such as movement, as discussed in this article. In that case, support sub-nets can be coupled through shared places or synchronous transitions, allowing the control net to interact with an obstacle avoidance net.

As we can see, the UAV swarm can navigate through the mission's steps (*places*) using the developed control network. In addition, the check for transition firing possibility allows us to ensure that UAVs in the swarm are moved only from one valid state to another and can be safely brought back in the event of a malfunction. Additionally, if some UAVs are lost, the mission will continue until at least one operational UAV remains, which enhances reliability.

5. Performing testing of the developed UAV swarm control model based on Petri net

The investigation was performed on the prototype HLPN model described in Section 4.2. Two Petri net variants were examined: one without the auto-return transition and one with it enabled. The experiment proceeded as follows:

– Scripts were prepared to generate random mission profiles (launch charge 70–90%, route length 5–10 km) and to execute both nets in the Snakes engine.

– A total of 100 UAVs were instantiated; their battery levels, routes, and completion statuses were logged.

– Every 20 milliseconds, the simulator updated token markings and wrote results to a CSV file, enabling live plotting.

The scatter plots on Figure 2 shows the success rate with and without auto-return. The UAVs in the bottom-right corner (those that have the longest route and lowest charge) have exhausted their batteries and are marked with red cross symbol, according to (3).

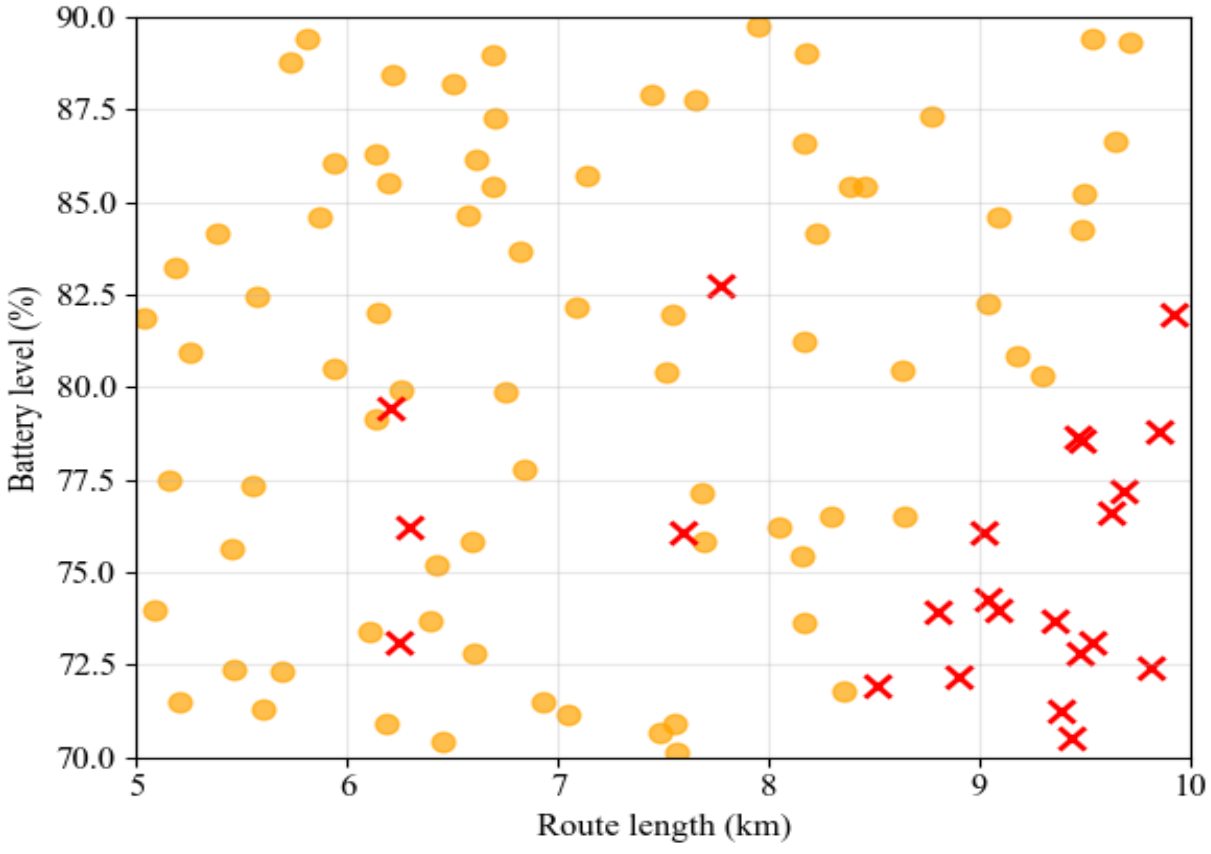

Fig. 2 (*a*). Number of failed UAVs: model without auto-return

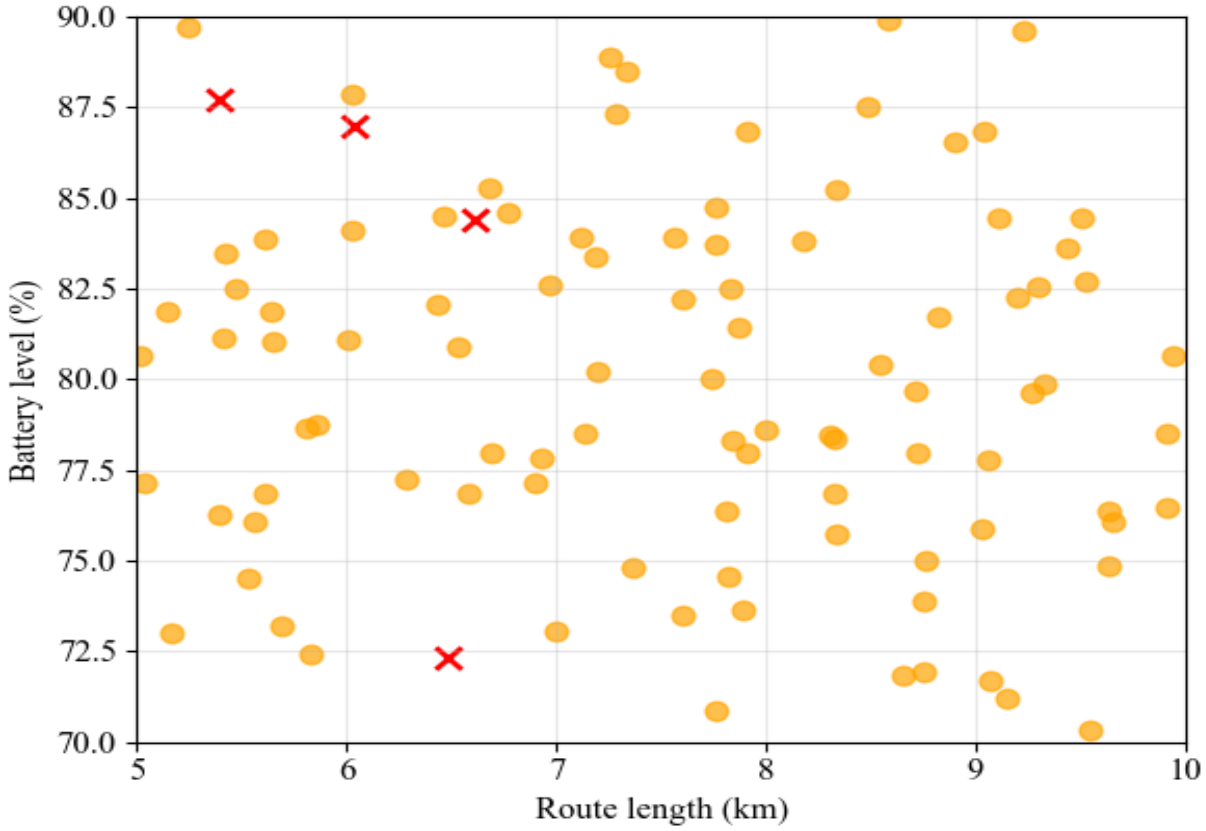

Fig. 2 (*b*). Number of failed UAVs: model with auto-return

Figure 2*a* presents the plot without auto-return step and 2*b* repeats the plot with auto-return; on the second plot the failure cluster disappears, confirming that the transition prevents energy depletion.

The experiment demonstrates that adding guard-driven transitions to the HLPN eliminates mission failures under the given energy model and parameter set. In other words, with this new check, the swarm controller can check that for some UAVs, there is not enough power and set it in auto-return mode. Some UAVs still fail due to other reasons, such as connection issues or other technical problems.

6. Discussion of the results from the development of the HLPN model

The formal reachability analysis conducted in Snakes confirms that the HLPNs controller fulfills its three target properties: the UAV never departs *Ready* with insufficient energy, every legal mission eventually reaches *ReturnToBase*, and the battery charge remains within physical bounds. This result is significant because it is obtained directly from the net's mathematical semantics rather than from long-running simulations. In practice, it can be rerun on every hardware whenever we tweak a guard or add a new flight stage, delivering continuous assurance at design time.

The UAV’s structure is a single-colored token that carries battery, position, and mission flags. This fact keeps the diagram readable while avoiding the need for auxiliary scripts. Guard predicates are expressed in ordinary Python using Snakes library. It makes domain rules transparent to both avionics specialists and software developers, easing code reviews and audits. Also, the simple checks can be extended to more advanced, for example they can be using machine learning models instead of simple boolean functions while still using Python only tools. Moreover, these checks in future can use monitoring data from real hardware without significant changes to the model. Because Snakes executes the same HLPN that the analyzer verifies, there is no semantic gap between the specification

and the runtime artifact; this “executable specification” property is a marked advantage over behavior trees or ad-hoc state machines, which often drift from their documentation.

Nevertheless, the current evidence is limited to *model-in-the-loop* verification. No hardware-in-the-loop or field tests were performed, so the impact of actuator latency, high-frequency sensor noise, and wind-induced load remains unqualified. Guards that rely on accurate battery estimates or GPS data will need runtime margins to withstand real-world uncertainty. The model also treats environmental hazards, such as sudden geo-fence updates or collision avoidance, as external events without probabilistic timing. Extending the HLPN with timed or stochastic transitions is a logical next step. Additionally, the steps are relatively simple, but in the future, it will be easy to integrate advanced scenarios for states like *Landing* [11].

Finally, while the Snakes tool chain integrates smoothly with Python-based simulators, teams already invested in GUI-oriented editors, such as CPN IDE, may face a learning curve when migrating to a code-centric workflow. A thin visualization layer that renders live markings back to a graphical view could mitigate this issue and broaden adoption among non-programming stakeholders. Also, created model can incorporate AI-based enhancements, such as reinforcement learning to optimize swarm strategies, without violating formal guarantees.

In summary, the proposed HLPN model delivers machine-checkable safety guarantees with modest computational effort and offers a clear upgrade path over traditional mission scripts. To translate these theoretical gains into operational reliability, future work should couple the model with real-time telemetry, execute flight-hardware-in-the-loop tests, and refine guard margins against empirical flight data.

7. Conclusion

In this study, we developed a formalized control model for UAV swarms based on HLPN that addresses the issues of adaptability, safety, and fault tolerance in dynamically changing environments. By representing the behavior of each UAV as a bipartite Petri net with typed tokens, we achieve a unified description of flight phases, resource constraints, and mission logic.

The main result includes a scalable HLPN based control system model that aggregates individual UAVs as *Tokens*. It supports energy-efficient and fault-tolerant mission coordination tasks. Because of using pure Python library (Snakes) as base, mission steps can be easily added or modified without completely rewriting the model, allowing to expand the model for more complex scenarios regardless of the number of drones. We also developed formal safety rules and transient protections that ensure that UAVs operate only under permissible conditions, preventing dangerous states (e.g., insufficient battery power for takeoff) and guaranteeing the reachability of mission objectives (e.g., final return to base). After these safety rules were introduced, the UAVs are not able to get in invalid state because the mission aborts immediately and they are automatically returned back to base. It increased success rate from 76% to 95%. The remaining 5% failed because of unpredictable reasons that always exists in dynamically changing environments.

The implementation proposed model allows reachability analysis, what-if testing, and has sequence logging, providing a solid foundation for evaluations.

The proposed HLPN-based approach switches UAV swarm management by replacing ad-hoc logic and manual checks to a mathematically sound, proven safe, and adaptive mechanism. It can be applied to both civilian and military scenarios, including surveillance, search and rescue, and delivery missions, where autonomous decision-making and reliability are critical.

References

[1] *ISO/IEC 15909-1:2019 – High-level Petri nets – Part 1: Concepts, definitions and graphical notation*, International Organization for Standardization, Geneva, Switzerland, 2019

[2] R. David, H. Alla, *Discrete, Continuous and Hybrid Petri Nets,* 2nd ed. Berlin, Germany: Springer, 2010, pp. 337

[3] S. Byun, D. Lee, “Performability Evaluation of Autonomous Underwater Vehicles Using Phased Fault Tree Analysis”. *Journal of Marine Science and Engineering*. vol. 12(4) pp. 564–575, 2024, https://doi.org/10.3390/jmse12040564

[4] A. Fedorova, V. Beliautsou, & A. Zimmermann. “Colored Petri Net Modelling and Evaluation of Drone Inspection Methods for Distribution Networks”. *Sensors*, vol. 22(9), pp. 3418–3438, 2022, doi: https://doi.org/10.3390/s22093418

[5] P. Bizhao, H. Xinting, D. Wei, H. L. Kin, “UAV path optimization with an integrated cost assessment model considering third-party risks in metropolitan environments”. *Reliability Engineering & System Safety*, vol. 222, Art. no.108399, 2022, https://doi.org/10.1016/j.ress.2022.108399

[6] H. Qin, B. Zhao, L. Xu, X. Bai, “Petri-Net Based Multi-Objective Optimization in Multi-UAV Aided Large-Scale Wireless Power and Information Transfer Networks.” *Remote Sensing*, vol. 13(13), pp. 2611–2630, 2021, https://doi.org/10.3390/rs13132611

[7] X. Wang, Y. Guo, N. Lu, P. He, “UAV Cluster Behavior Modeling Based on Spatial-Temporal Hybrid Petri Net.” *Applied Sciences*, vol. 13(2), pp.762–778, 2023, https://doi.org/10.3390/app13020762

[8] Gazebo, “Getting Started with Gazebo”, gazebosim.org, Accessed: May 6, 2025. [Online] Available: https://gazebosim.org/docs/latest/getstarted/

[9] C. Brito, L. Silva, G. Callou, T.A. Nguyen, D. Min, J.-W. Lee, F.A. Silva, “Offloading Data through Unmanned Aerial Vehicles: A Dependability Evaluation.” *Electronics*, vol. 10(16), pp.1916–1934, 2021, https://doi.org/10.3390/electronics10161916

[10] M. A. Trigos, A. Barrientos, J. del Cerro, “Systematic process for building a fault diagnoser based on Petri nets applied to a helicopter.” *Mathematical Problems in Engineering*, vol. 1, 13 p., 2015, https://doi.org/10.1155/2015/963756

[11] A. Bertolaso, M. Raeissi, A. Farinelli, R. Muradore, “Using Petri Net Plans for Modeling UAV–UGV Cooperative Landing” *Frontiers in Artificial Intelligence and Applications* vol. 285, pp.1720–1721, 2016, https://doi.org/10.3233/978-1-61499-672-9-1720

УДК 004.67, 004.38

МОДЕЛЬ КЕРУВАННЯ РОЄМ ДРОНІВ НА ОСНОВІ ВИСОКОРІВНЕВИХ МЕРЕЖ ПЕТРІ

Валентин Іванков
Національний Технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
https://orcid.org/0009-0001-9835-8486

Михайло Новотарський
Національний Технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
http://orcid.org/0000-0002-5653-8518

Швидке зростання опцій застосування безпілотних літальних апаратів (БПЛА) у сучасному світі висуває доволі серйозні вимоги до надійності логіки керування. Помилка в послідовності стадій загрожує як мінімум нераціональним використанням батареї або порушенням повітряних правил, а як максимум то аварією з втратою апарату і заподіяння шкоди. Зазвичай керування будується на скриптах або поведінкових деревах, що ускладнює супровід. Причина в тому що розмір вихідних файлів швидко зростає, і при необхідності додати новий функціонал чи модифікувати існуючий, виникає загроза внесення певної вразливості через не врахування всіх можливих ситуацій. Саме тому було обрано високорівневі мережі Петрі, оскільки цей метод вирішує проблему формального опису системи керування, а також дозволяє легко масштабувати дану систему або вносити в неї будь-які зміни.

Метою дослідження є розробка та валідація моделі на основі високорівневої мережі Петрі, яка буде виступати як джерело істини для керуванням роєм БПЛА. У запропонованої моделі, стани відповідають етапам польоту, а токени несуть числові параметри, такі як заряд акумулятора, координати, телеметрія. Таким чином одна схема одночасно описує дискретні події та обмеження. Для кожного переходу формалізовано умови, що перевіряють можливість його здійснення, наприклад перевірка мінімально необхідного заряду батареї або перевірка локації. Методологія включає декілька етапів. Перший це формальне визначення структури мережі. Далі на основі структури будується модель з використанням мови *Python*, яка реалізує розроблену мережу, контролює переміщення між станами та правильну послідовність спрацьовування переходів. Після розробки моделі виконується тестування і аналіз отриманих результатів.

Результати показують, що використання високорівневих мереж Петрі для побудови моделі перевірки команд у дискретному режимі справді забезпечує коректний опис роботи переходів між станами та підвищує надійність і живучість розробленої моделі системи керування, а також значно скорочує зусилля на підтримку. Розроблена модель легко адаптується до змін маршруту, додавання датчиків, або розширення функціоналу.

Ключові слова: високорівнева мережа Петрі, безпілотний літальний апарат, симуляція, система керування.