AIAA SciTech Forum
6-10 January 2025, Orlando, FL
AIAA SCITECH 2025 Forum

10.2514/6.2025-1356

Integrated Prognostics and Health Monitoring Tool for Software Components Aboard UAS Swarm Agents

Landen Fogle*, Grant Phillips†
University of Nebraska-Lincoln, Lincoln, NE 68588, USA

Justin Bradley‡
North Carolina State University, Raleigh, NC 27695, USA

Uncrewed Aircraft Systems (UAS) are pivotal in numerous fields, requiring dependable software architectures that reinforce functionality and efficiency. However, effective in-flight monitoring of these agents is often limited to verifying hardware performance and may lack monitors for more complex software systems. The problem is seen in small UAS multi-agent systems and swarms where bandwidth is minimal and computational resources are highly constrained. Here we introduce the development, processes, and evaluation of a Health Management and Control tool tailored for monitoring the health and operational status of essential UAS software architecture components. This tool facilitates system debugging and enhances operational efficiency through diagnostics and recovery-focused health management.

I. Introduction

Decentralized uncrewed aircraft system (UAS) swarming is an approach to enhancing mission capabilities through the use of several intelligent, independently-acting agents. In a decentralized system architecture, control, planning, and decision-making tasks are delegated to the agents towards improving the overall system robustness at the expense of increasing the complexity of the agent's software and hardware architectures. Inevitably, exercising a complex system that is under development will invoke failure cases, either as a result of design oversights or wear-and-tear on the hardware. Isolating the source of failures within a swarm is typically done by manually reviewing agent logs and field notes, or re-creating failure scenarios; however, these heuristic methodologies are time intensive and do not guarantee that the source(s) of a particular failure will be discovered.

Fig. 1 UAS swarm during field experimentation using a novel, decentralized controller.

To illustrate, consider an operator attempting to relocate a flying swarm to a new ground target. A command is sent from a ground control station (GCS) over a radio link to the swarm agents; the receiving radio is connected to an onboard companion computer and the command interpreted by the controller, see Figure 2. If some subset (but not all) of the agents fail to relocate then we need to uncover why some worked and some did not. From experience, possible causes of this failure could span a wide variety of potential cyber-physical issues: lost sent packets, radio disconnect/failure, the companion computer losing connection to the radio, logs on the companion computer have filled the free disk space, the companion computer ports have failed, or an uncaught exception has caused the command interpreter to crash. While manually isolating the failure is possible, significant time may be needed diagnosing all failing agents due to a broad range of failure sources. Furthermore, this is likely impossible during the flight when

*Undergraduate Student, UNL School of Computing, AIAA Student Member
†Graduate Student, UNL School of Computing, AIAA Student Member
‡Associate Professor, Department of Computer Science, AIAA Associate Fellow

the error presents. This example highlights the need for an automated methodology to quickly identify and anticipate sub-system failures within swarms.

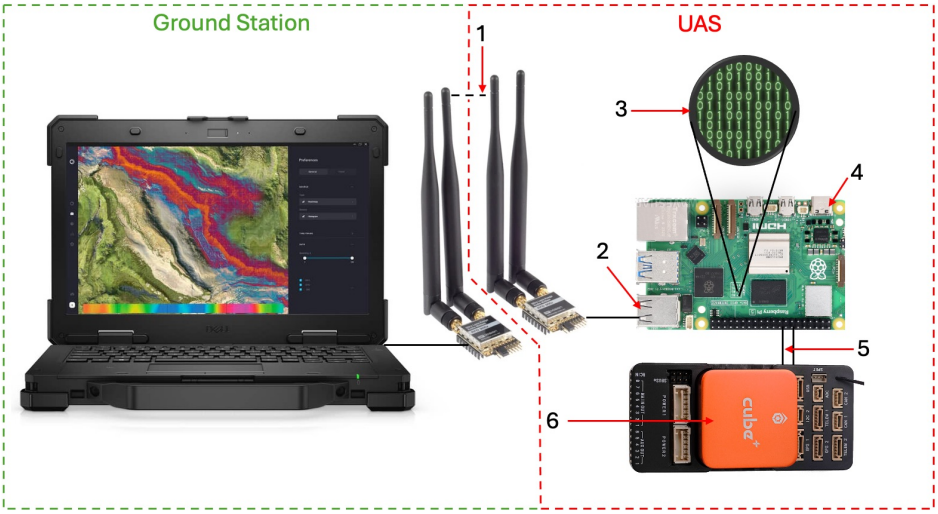

Fig. 2 A command sent from the ground station must "move" through connected hardware components to reach the flight control unit (hardware is described in Section III.A). The command encounters several points of failure: 1) the radio link, 2) the radio-companion computer connection, 3) the software handling the command, 4) the companion computer, 5) the flight control unit-companion computer connection, 6) the flight control unit. A failure at any of these stages will result in the command not being executed, however, the "location" of the failure would be unclear without further health monitoring capability.

Prognostics and Health Management (PHM) is an engineering practice that focuses on predicting the reliability and health state of complex, safety-critical systems. This methodology has been widely used by the aerospace, automotive, and manufacturing industries due to its ability to reduce system life-cycle costs. Life-cycle costs are reduced by adopting condition-based maintenance practices in place of traditional preventative and corrective maintenance strategies [1].

Most often, PHM is used to predict hardware component failure by supplying real-time sensor data to a system model that can distinguish nominal system performance from behavior that indicates a component is approaching failure. Several types of models have been explored, simple systems can be effectively monitored using regression methods while more complex systems have been modeled using neural networks and support vector machines [2].

In this work, we seek to extend the methodologies of PHM to monitor the hardware, but more importantly, the software health of a distributed, multi-agent UAS swarm composed of identical quadrotors. Distributed UAS swarms are characterized as a group of four or more "agents" that work cooperatively with one-another to achieve a common goal without a centralized controller or planner [3]. In this control scheme, each agent governs its own planning, decision, and control with limited communication between agents. Although the control complexity and computational burden is increased at the agent level under distributed control, the swarm achieves a much higher level of intelligence and autonomy, making it highly robust to individual agent failures, and thus increasing the likelihood the mission objectives will be met.

Monitoring the agents' health (from a software and hardware perspective) and long-term physical condition quickly becomes difficult as the number of agents in a swarm grows. The quadrotor hardware (e.g., motors, electronic speed controllers, sensing modules) are all single points of failure in the system and tracking flight hours, repairs, and mean-time-to-failure across the entire fleet is tedious for small research teams; moreover, crashes introduce uncertainty into these efforts and potentially reduce the expected life of parts. When a hardware failure does occur, locating the source of the failure requires several hours of sifting through vehicle logs (just for one vehicle). Furthermore, fielding experimental software introduces other failure risks that are time consuming to track down. Given the number of swarm agents, automated, on-board health-checking is needed to quickly and succinctly identify software and hardware failures. This paper will make the following contributions:

- A highly dependable and scalable tool created to monitor health check metrics aboard an agent.
- Provide health reports of swarm research flights pinpointing communication and software failures.

- Demonstration of accuracy and deficiency metrics on Software in the Loop (SITL) and hardware experiments.

II. Related Work

Our work stems off of a new engineering discipline in the industry, known as PHM. Related work in this field looks specifically at system health metrics during field operations and aims to create recovery strategies based on the health states. Many work combines the practices of fault prediction and diagnosis, and health management to ensure reliability where PHM practices are implemented [4].

PHM methods have been adopted across various industries due to their ability to improve the reliability of complex systems under variable and challenging conditions. One industry example of this practice is in marine engines. PHM concepts have been implemented in this space because of the complexities of marine diesel engines that power most commercial vessels. Because of their critical status in operation, health monitoring techniques like PHM have been crucial to their developmental practices. [5].

PHM techniques have also been seen in aerospace fields to increase the reliability of aircraft systems. Advances in data analytics and machine learning have propelled the development of these predictive maintenance strategies. This industry has benefited from this semi-supervised anomaly detection techniques and research applied to various components, of which the most popular are aircraft cooling units. These techniques utilize deep neural networks to detect anomalies from nominal operational data, thus allowing for early detection of potential aircraft failures. Importantly, the ongoing research demonstrates the potential of semi-supervised learning models to adapt to the high uncertainty. This would have the potential to reduce unscheduled maintenance costs and improve system reliability [6].

Robotics has also seen the integration of PHM, particularly when using the industry standard Robot Operating System (ROS). ROS* is an open-source framework that houses many tools and software libraries for developing robotics systems. ROS aids developers in both academia and industry environments, with creating anything from drivers to complex algorithms using a full software development kit (SDK). PHM tools within ROS are designed to enhance the reliability of any type of robotic systems, including mobile robots and drones. Current PHM tools are implemented to check for system health issues, health monitoring, estimation of remaining useful life (RUL), and prediction of task completion probabilities. [7].

The application of PHM in UAS forms one critical area of research with potential applications, especially in developing mission reliability and operational efficiency. UAS benefit from Prognostic Decision-Making (PDM) systems that utilize continuous Partially Observable Markov Decision Processes (POMDPs)[8]. These frameworks involve complex simulations toward managing non-linear degradation processes, besides the uncertainties associated with state estimation and the effects of the actions. This, in turn, enables UAS to dynamically re-plan the mission if component faults are detected during flight, which will significantly increase the success rate. These are enabled by advanced randomized algorithms supported by detailed physics simulators to generate and evaluate potential future states and actions to ensure reasonable operations under unpredictable conditions [9].

Further integration of PHM in UAS is observed through the use of Integrated Vehicle Health Management (IVHM) techniques for assessing the RUL of UAS components. Such techniques often involve fault tree analysis fed by probability density functions, which that dynamically adapt to the aircraft health state and therefore limit uncertainty during flight missions. [10] The recent interest in deep learning applications has further pushed the adaptation of machine learning-based frameworks in anomaly detection for UAS. These studies typically utilize clustering algorithms to effectively label flight data, and thus allow the detection of potential anomalies before they affect system performance. This intersection of machine learning and PHM tools represents significant developmental progress and pivot to predictive analytics approaches in UAS health management [11, 12].

Current research in the area of PHM, especially within UAS operations, has predominantly concentrated on enhancing anomaly detection capabilities. These methods use new techniques including machine learning models or through the application of non-linear decision optimization algorithms. These methods typically focus on the robust nature of custom hardware or the monitoring of system functionalities [8]. However, a notable limitation of these approaches is their heavy reliance on predictive analytics, which while powerful, may not fully address all operational contingencies. Especially in the UAS space, these heavy weighted tools do not allow for systems working with limited resources These approaches are not only computationally straining, which affect efficiency, but also limited by necessary hardware onboard the agent.

Contrary to the common trends in PHM research, our project adopts a different approach by developing a PHM software tool designed specifically for reporting the performance and uptime of UASs. This tool is unique in that

*https://www.ros.org/

it does not rely on machine learning algorithms or predictive outputs, which are not only common in the field, but also resource intensive. It additionally does not require high levels of bandwidth in turn having minimal effect on communication networks and hardware aboard the UAS. This approach is more suitable in small UASs due to their limited computational and power resources. Instead, our focus is on direct reporting to ground stations to address discrepancies in communication which is a frequent issue in UAS swarm operations. We are creating a tool that is tailored to detect and log specific types of communication failures, such as disruptions in radio transmissions, issues within Raspberry Pi hardware, malfunctions in Docker container processes, or failures in ROS nodes. By identifying the locations of potential communication failures between what is sent to the agent and the outcome, we aim to immediately alert the ground station control (GSC) and minimize downtime for any operational swarm team .

III. Methodology

The core objective of this research is to develop an extendable methodology that helps quickly and accurately identify software failures during swarm missions. This work focuses specifically on the health of individual agents within the swarm, and then creating a summary report of the entire swarm. This involves a straightforward approach to testing and monitoring each layer of the software stack. The software stack for this research includes three primary layers: Companion Computer, Docker, and ROS layers - visualized in Figure 3. This layer structure will ensure full coverage by the PHM tool from the hardware layer up through the higher-level application layers. Each of these layers has its respective monitoring metrics and specific monitoring and recovery mechanisms.

A. System Overview

In this section, we provide an overview of a swarm agent's hardware and software components to better contextualize our methodology; these are illustrated in Figure 3. While the system described is our specific implementation, the core components (autopilot, companion computer, radios) are representative of other intelligent UAS systems.

Our UAS agents utilize an off-the shelf pixhawk-style autopilot (Cube Blue) with either PX4 or ArduPilot firmware. The autopilot is a self-contained embedded system that manages the UAS's internal inertial measurement units, gyroscopes, barometers, and GPS modules as well as realizing a desired roll, pitch, and yaw, stabilizing the vehicle, or navigating to global waypoints. These off-the-shelf components are supplemented by a companion computer (Raspberry Pi/Intel NUC/Nvidia Jetson) that executes other control or computation beyond the autopilot such as high-level autonomous planning, image processing, or in our case a cluster of swarm control processes (nodes). The companion computer and autopilot communicate via MAVLink[†]: a lightweight messaging protocol designed for micro aerial vehicles. Through MAVLink, the autopilot sensor readings are available to the companion computer, and the companion computer can send control commands to the autopilot. The swarm agents are connected via two separate networks: 1) a 2.4 GHz asynchronous inter-agent XBee network used to exchange agent states and 2) a 900 MHz synchronous agent-to-GCS RFD900 network used for slower, long-range backup commands.

Our custom swarm planning and control applications [13] are implemented on the companion computer within the ROS framework. ROS provides a relatively simple means to develop and run processes that interact with sensors and actuators through publisher-subscriber messaging channels. While ROS offers the means to easily implement our swarm planning and control applications, managing the ROS application and software environments (i.e., code changes, operating system, device drivers, network mapping) across several companion computers is tedious and if not properly managed, can lead to failure. To eliminate the adverse effects of mismanaged software/software environments, we have adopted the Docker dev-ops paradigm in which the application *and* the environment is packaged in a pre-built image/container that can be easily transported between companion computers.

B. PHM Tool Architecture

Similar to our ROS swarm control application, the PHM software tool is deployed within a separate Docker container on the companion computer so that the tool can be readily used on any operating system that supports the Docker Engine. This container utilizes a common docker feature known as volume mounting to enable data sharing with other containers, or the container host. Volume mounting allows containers to access and share files without altering the data permanently. This enables the PHM container to access files relating to ROS logs and outputs of the ROS application container. This setup is necessary for performing health checks and diagnostics without disrupting the swarm ROS packages. Next, we will break down the health monitoring at each system layer, as depicted in Figure 3.

[†]mavlink.io

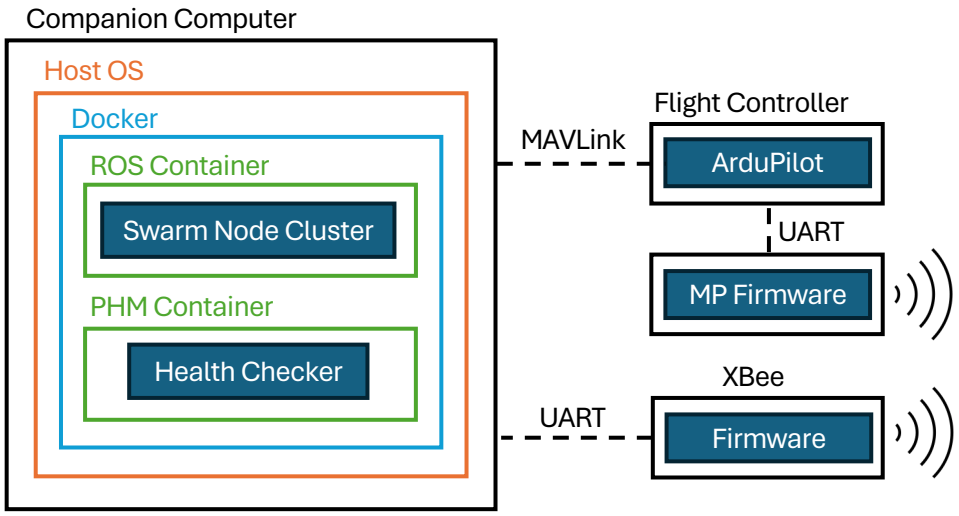

Fig. 3 Representation of a swarm agent's hardware and software organization. The PHM container communicates and shares files with the ROS container via shared network and volume respectively. MAVLink is used to facilitate communication with external swarm agents and the GCS.

1. Companion Computer Health Monitoring

The initial health monitoring layer of our PHM framework is directed towards the companion computer aboard each agent within the UAS swarm; which is typically running a version of Ubuntu on hardware such as a Raspberry Pi or a similar low-level device[‡]. The companion computer acts as the central processing unit, orchestrating the functionality and managing the data critical to each UAS. This top-level layer is key to the operational health of an agent. Here, the companion computer regularly sends heartbeat signals to the GCS at set intervals. These signals confirm the operational status of the UASs, thus ensuring communication between the agents and the GCS is functional. This method facilitates early detection of communication failures or hardware issues.

2. Docker Health Monitoring

Each agent's companion computer hosts Docker containers that encapsulate the software necessary for swarm mission execution. Docker is often used as a container service because of its wide array of benefits, primarily its consistency across multiple environments to reduce environment setup and conflicts. Because this research is focused on maximizing efficiency and minimizing resources, Docker's ability to create lightweight containers allows us to focus on the deployed code and resources. In small UAS, hardware, and environmental resources are often limited, making this a large benefit of using Docker. Another advantage is Docker's ability for repeatable and scalable software deployment. Since we are working with large swarms of identical copies of the same hardware and software, this allows for easy deployment. Containerizing these applications ensures developers can roll out updates seamlessly to all swarm agents.

For portability, health checks are carried out at the Docker container level and are designed to test three essential aspects of swarm UAS operations: 1) ensuring the presence and validity of operational files within each container 2) verifying the continuous connectivity of each agent with the application container 3) validating connected devices. Standard practice within Docker, Kubernetes, or any container-based service is to implement two types of probes: liveness and readiness. Liveness probes help determine if the container is active and running as expected, while readiness probes ensure the container is fully prepared to handle communication tasks with the GCS from internal software. Additionally, connected devices to the swarm agents are critical and often cause failures if not connected properly. To include this in our health monitoring, device ports accessed by the Docker container will be checked for discrepancies from what is expected. The failures within the Docker layer will result in logging, GSC notifications, and Docker container reboot. Additionally, monitoring will be in place to read Docker's embedded 'Healthcheck' property to provide a continuous review of the application container status. One of the key features of our tool is the automatic restart when failures are detected. This automated recovery system is crucial to minimize our operational downtime for

[‡]Raspberry Pi's are often favored in UAS applications due to their balance of computational capability, low power consumption, and integration with open source platforms like ROS.

the application container.

3. ROS Health Monitoring

The ROS layer facilitates communication and operational consensus among various software nodes built into our Docker containers as seen in Figure 3. Health checks performed on these nodes broadcast and receive signals (Pub-Sub Model) to verify the existence and performance of adjacent nodes. This inter-node communication facilitates its consensus mechanism illustrated in Figure 4. As seen in the figure, the consensus on node status is communicated to a specific topic, that in turn the PHM tool monitors. This is considered more reliable rather than directly interfacing with the GCS. This tool can be dynamically configured to adjust to the number and specific functions of ROS nodes it monitors, thereby providing a scalable solution adaptable to varying operational needs within the UAS swarm. The PHM tool ensures that each node is not only active but also effectively participating in the network's communication by monitoring the active topic connections. This comprehensive monitoring of topic traffic is the PHM tools process for detecting any discrepancies in communication between the agent and the GCS.

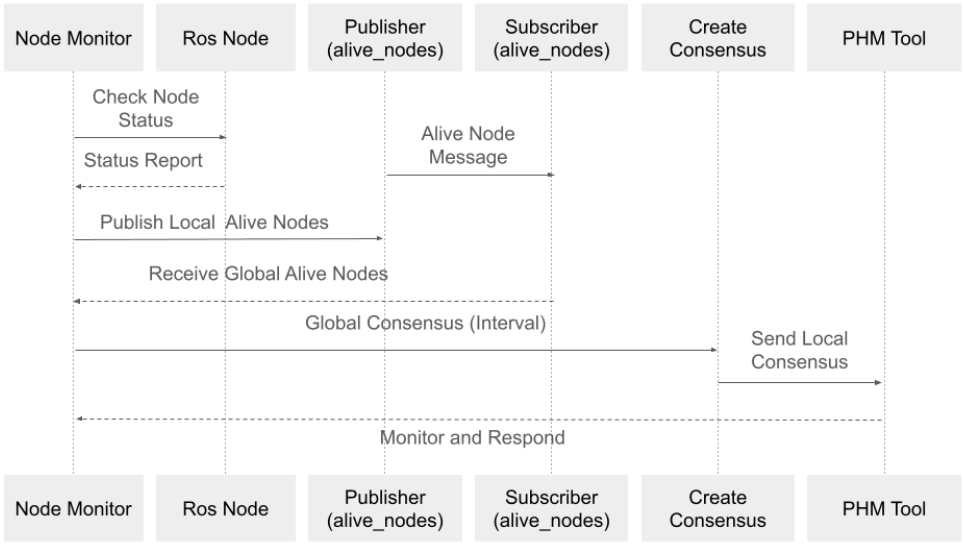

Fig. 4 Sequence diagram of ROS node consensus mechanism.

After the PHM tool has successfully monitored the application container, the container digests the logs created in two primary methods. First, logs are maintained within the Docker container, which contains more detailed outputs of the issues detected by the PHM tool during swarm flights. These logs serve as a permanent record that can be reviewed for troubleshooting and system refinement. Secondly, the GCS server is configured to receive a summary of the monitored data. This summary will be in a high-level form of what agents are failing and at what level. This dual approach not only ensures an option for immediate awareness at the ground level, but also more detailed operation data post-flight for inspection and after-the-fact enhancements. Once the container has completed its reporting it waits for a reactivation interval that is provided within the environment before reassessing the application container again.

C. Testing

We have validated the PHM tool's functionality using a simulation built around ArduPilot's[§] Software in the Loop (SITL) capabilities. SITL provides a controlled, repeatable, and safe testbed where software systems can interact with virtual hardware to simulate flight dynamics and operational scenarios that are uniform with the real deployment environment. Multiple SITL instances are locally networked via a MAVLink router that re-creates the wireless mesh-network used in real deployments. This simulation methodology is particularly beneficial for early-stage testing of software tools like our PHM system because it allows developers to introduce and precisely control fault conditions.

Our validation approach consists of artificial "attacks" to the simulated system in the form of various faults within

[§]https://ardupilot.org/

the SITL environment. This methodology involves deliberately introducing different faults into the system over multiple trials, to observe how the PHM tool detects and responds to these disruptions. We created a manual testing suite of functions to deliberately kill parts of the architecture from Figure 3. The faults are designed to test each layer and health measure outlined in Table 1. The SITL environment facilitates both single and multiple attack trials, allowing us to observe the behavior of the PHM tool under isolated as well as compounded fault conditions.

Table 1 Health Checks Performed by the PHM Tool

Layer	Specific Check	Description	Method	Response
Companion Computer	Heartbeat Status	Checks if the companion computer is responsive.	Periodic heartbeat checks.	Alert and log the status; initiate troubleshooting if unresponsive.
Container Service	Correct and Executable Files	Ensures that all files within the container are present and executable as expected.	Docker Health Check Flags with File integrity scans.	Alert and Log
	Network Communication Capabilities	Assesses the ability of the containers to be connected to the same network.	Docker Health Check Flags with Network connectivity tests.	Alert and reconnect to shared network
	Port Devices Connected	Verifies that all required devices are correctly connected and recognized.	Docker Health Check Flags with Device connection verification	Alert and Log
ROS	Consensus of Running Nodes	Ensures that all active ROS nodes are in consensus regarding system status and operational parameters.	ROS topics.	Alert and Log
	Node Topic Connections	Checks that each ROS node is properly connected to necessary topics for communication.	Topic subscription audits.	Alert and Log
	Topic Traffic Monitoring	Monitors the flow of messages across topics to ensure fluid communication between steps.	Traffic analysis and integrity checks.	Alert and Log

IV. Results

The main software artifact from this research is our containerized prognosis and health management tool. The tool periodically tests other local application containers on each swarm agent and generates health reports containing detailed insights into the status of the individual agent. Upon gaining status it is then broadcast out for other agents to listen and build consensus through the swarm. Ground stations receive updates through the MAVLink infrastructure informing the swarm operators of the status of the agent. The tool has been designed to be lightweight and scalable ensuring compatibility and integration into current swarm architectures. The tool is designed to inform users of potential faults in different layers of the system allowing users to know where packets of information or requests are failing. Summary reports inform users of where the fault may be occurring and if the system will be able to recover autonomously. Particular faults are able to be fixed in real time as the container provides actionable responses to a third of the pre-designed health checks. If more detailed information is needed, logs are available on agents for post flight analysis.

A. Performance and Evaluation

The PHM tool was evaluated by performing SITL simulations with different agent environments. "Faults" were injected into our simulation to test each of the predetermined health checks. Overall testing resulted in high performance metrics and high detection rates for all pre-defined health checks. As shown in Table 2, the parameters used for the tool during testing were the number of agents in the SITL environment and the reactivation interval. The reactivation interval was set to 20 s to balance the additional radio bandwidth consumed by PHM monitoring with timely fault detection. Additionally, 8 SITL agents were used in our testing runs, this number of agents reflects the deployments from [13]. Overall the trials of the tool showed strong performance metrics including high uptime and accuracy, leading to many injected faults properly documented, logged, and mitigated properly.

Table 2 PHM Tool Performance Metrics

Metric	**Value**
Parameter: Reactivation Interval	20 s
Parameter: SITL Agents	8
Average PHM Response	14.82 s
Average Fault Detection	5.2 s
Self-Resolved Faults	16.6 %
PHM Uptime	96.77 %

The results outlined in Table 2 are the result of twenty SITL simulations following the testing plan outlined in section III.C. The average PHM response time corresponded to the tools ability to publish a summarized report from its previous testing suite. This metric corresponds to the interval in which other agents are informed of that status of its peers. The average fault detection rate was calculated by taking the average time from fault injection until the tools complete response and logging of the fault. Additionally measures were calculated from logs similar to Figure 5, including uptime of the containers and the number of faults resolved by the PHM tool.

B. Limitations

The purpose of this research was to explore a lightweight and easily deployable methodology to identifying where faults may be occurring on intelligent agents in a swarm. This preliminary research was conducted in a proof of concept way which introduces a handful of limitations that can be solved with continued development. The tool's primary limitations are bi-products of our software design. Currently checks often are prone to failure based on their dependency lists. Health checks are often dependent on previous checks testing layers above them respectively. This often would result in a chain of errors during testing that were not necessarily inaccurate however the result of too tightly coupled design. Indirect to the results, the container should be re-factored to follow better software design principles including class hierarchies, interfaces, and encapsulation to limit future limitations.

Hidden limitations stem from relying solely on SITL testing in this research. Hardware-in-the-loop or live field testing could have uncovered additional constraints within the system. Hardware testing introduces additional issues such as hardware specification limitations, possible electrical and power failures, and direct radio bandwidth constraints. These in turn create possible communication bandwidth challenges, especially in environments with limited radio frequency availability or significant interference. The scale of the swarm was tightly mimicked in SITL testing to try and identify these constraints; however, due to resources, these limitations are still unknown. To address these constraints, proposed solutions include compressing health check data to reduce transmission size and optimizing the frequency of health reports. These improvements would enable swarms to scale horizontally while maintaining the use of the PHM tool across agents.

Additionally, due to the initial health checklist, faults outside of these outlined health checks will go unnoticed unless they affect the monitored system. Although the goal of the tool was to be lightweight, non-predictive, and niche-focused, this often was found to be a limitation when testing the health of the system overall. Expanding the PHM tool to support dynamic or user-defined health checks through a graphical user interface would enable the tool to adapt to new fault scenarios without requiring manual updates. This tool serves to solve the problem of identifying and monitoring the system's ability to perform the mission provided, by ensuring the health of the system-level components aboard the agent.

```
2024-12-03 03:34:14 ---- Start of a new report
2024-12-03 03:34:22 INJECT Command executed successfully: killing /agent_1/node_1
2024-12-03 03:34:34 ---- SUCCESS: (1) Companion Computer is running aboard agent.
2024-12-03 03:34:34 ---- DOCKER: Checking health of ros_app_container_1
2024-12-03 03:34:35 ---- ACTION: Connected volume /home/**/AIAA_PHM_Tool/ros_app/ros_packages
2024-12-03 03:34:35 ---- ACTION: Connecting PHM tool to ROS app's network: phm_shared_network
2024-12-03 03:34:35 ---- SUCCESS: PHM tool successfully connected to phm_shared_network
2024-12-03 03:34:35 ---- SUCCESS: (2) Container health_management_container_1 is running
2024-12-03 03:34:35 ---- BYPASS: (2) No health check defined for health_management_container_1.
2024-12-03 03:34:35 ---- SUCCESS: (4) Both containers are connected to the same network: phm_shared_network
2024-12-03 03:34:35 ---- SUCCESS: (4) IP Ping network check passed for ros_app_container_1
2024-12-03 03:34:35 ---- SUCCESS: (3) File check passed: /home/catkin_ws/src/critical_file.
2024-12-03 03:34:35 ---- SUCCESS: (5) Found 8 devices connected to container
2024-12-03 03:34:36 ---- FAILURE: (6) Not all required nodes are running. Missing nodes: agent_1/node_1
2024-12-03 03:34:37 ---- SUCCESS: (7) All required topics are present: agent_1/topic_1, agent_1/topic_2, agent_1/topic_3
2024-12-03 03:34:55 ---- FAILURE: (8) Some topics are offline. agent_1/topic_1: offline, agent_1/topic_2: 0.999, agent_1/topic_3: 1.001
2024-12-03 03:34:55 REPORT Successes: 6, Failures: 2, Status Array: [1, 1, 1, 1, 1, 0, 1, 0,]
```

Fig. 5 An example log file from a SITL simulation showing the results of a fault injection test, where a command was issued to terminate a specific ROS node within the ROS application container. The test recorded six successful status checks and two failures directly linked to the fault injection event.

C. Practical Applications

Use cases of this tool are directly tied to decentralized swarm systems, especially those relying on the ROS framework for robotics applications. Its primary use case involves diagnosing and resolving existing faults rather than predicting potential issues, making it ideal for real-time fault management in operational environments. Unlike identifying existing solutions focusing on predictive and learning PHM algorithms, we focused on identifying existing faults and using that information to help operators make informed decisions about the root cause of the failure. The tool looks for faults along the systems that could possibly interfere with swarm commands. When using this tool swarms can expect to have low levels of bandwidth impact while receiving logs and in flight reporting on critical systems aboard the agents. This capability is particularly valuable in scenarios where an agent becomes unresponsive, as the tool can determine whether the issue lies in a recovery attempt, or complete mission failure.

This work is a step towards developing real swarm applications with a "swarm mentality". That is, the approach to handling swarms (debugging, maintenance, fielding) is very different than the approach used with 1 or 2 agents due to time and personnel limitations. Swarm developers must automate as many time burdens as possible, including answering the question: "What caused this agent to fail", and using this knowledge to remedy the failure. The methodologies and artifacts presented here fill another gap towards developing robust and reliable intelligent swarms.

V. Conclusions

A significant challenge in decentralized swarm deployments is the system complexity and emergent behavior that can arise from inadvertently violating compositionality and composability system design principles. This makes errors, bugs, and hardware problems **very** difficult to track down without more rigorous onboard health monitoring. Here we have described the foundation for an operational prognostics and health monitoring tool for robotic drone swarm deployments. We have explicitly focused on drone swarms that leverage a containerized ROS image to integrate the autonomy stack into the autopilot as this has suited our use case. A common failure point is various component, ROS node, and container failures – a problem the tool will notify users about in a timely fashion.

One drawback of customized tools of this nature is their often poor applicability in different systems, integrations, and deployment environments. To avoid creating yet another one-off tool not widely applicable, we are actively working to evolve our tool into a more flexible mechanism by which tests and checks can be easily developed and "plugged in" to the software framework. This will allow developers and integrators the ability to customize their failure checks, monitor health states of components, and alert users in the ways that work for their system while providing a set of backbone PHM services.

Acknowledgments

This work supported in part by NSF-2047971, USDA 2023-67021-38977, NSF-2221648, and NSRI FA4600-20-D-0003. This work was supported in part by AI for assistance in research, manuscript ideation, and figure creation.

References

[1] Gray, D., Rivers, D., and Vermont, G., "Measuring the Economic Impacts of the NSF Industry/University Cooperative Research Centers Program," 2012. URL https://www.slideshare.net/slideshow/iucrceconimpactfeasibilityreportfinalfinal-58582920/58582920.

[2] Kim, H.-E., Tan, A. C., Mathew, J., Kim, E. Y., and Choi, B.-K., "Machine prognostics based on health state estimation using SVM," *Asset condition, information systems and decision models*, 2012, pp. 169–186.

[3] Arnold, R., Carey, K., Abruzzo, B., and Korpela, C., "What Is A Robot Swarm: A Definition for Swarming Robotics," *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2019, pp. 0074–0081. https://doi.org/10.1109/UEMCON47517.2019.8993024, URL https://ieeexplore.ieee.org/abstract/document/8993024.

[4] Qiao, H., "Prognostics, health management, and Control (PHMC)," *NIST*, 2018. URL https://www.nist.gov/programs-projects/prognostics-health-management-and-control-phmc.

[5] Gharib, H., and Kovács, G., "A Review of Prognostic and Health Management (PHM) Methods and Limitations for Marine Diesel Engines: New Research Directions," *Machines*, Vol. 11, No. 7, 2023. https://doi.org/10.3390/machines11070695, URL https://www.mdpi.com/2075-1702/11/7/695.

[6] Basora, L., Bry, P., Olive, X., and Freeman, F., "Aircraft Fleet Health Monitoring with Anomaly Detection Techniques," *Aerospace*, Vol. 8, No. 4, 2021. https://doi.org/10.3390/aerospace8040103, URL https://www.mdpi.com/2226-4310/8/4/103.

[7] Gencturk, H., Erdogan, E., Karaca, M., and Yayan, U., "Prognostic and Health Management (PHM) tool for Robot Operating System (ROS)," *CoRR*, Vol. abs/2011.09222, 2020. URL https://arxiv.org/abs/2011.09222.

[8] Balaban, E., and Alonso, J. J., "A modeling framework for prognostic decision making and its application to UAV mission planning," *Annual Conference of the PHM Society*, ???? URL https://papers.phmsociety.org/index.php/phmconf/article/view/2294.

[9] Pan, J., Qu, W., Xue, H., Zhang, L., and Wu, L., "Study on fault prognostics and Health Management for UAV," *IOS Press Ebooks*, 2022. URL https://ebooks.iospress.nl/doi/10.3233/FAIA220569.

[10] Paixão de Medeiros, I., Ramos Rodrigues, L., Strottmann Kern, C., Duarte Coelho dos Santos, R., and Hideiti Shiguemori, E., "Integrated task assignment and maintenance recommendation based on system architecture and PHM information for UAVs," 2015, pp. 182–188. https://doi.org/10.1109/SYSCON.2015.7116749.

[11] Ahn, H., Choi, H.-L., Kang, M., and Moon, S., "Learning-Based Anomaly Detection and Monitoring for Swarm Drone Flights," *Applied Sciences*, Vol. 9, No. 24, 2019. https://doi.org/10.3390/app9245477, URL https://www.mdpi.com/2076-3417/9/24/5477.

[12] Ahn, H., "Deep Learning based Anomaly Detection for a Vehicle in Swarm Drone System," 2020, pp. 557–561. https://doi.org/10.1109/ICUAS48674.2020.9213880.

[13] Phillips, G., Bradley, J. M., and Fernando, C., "A Deployable, Decentralized Hierarchical Reinforcement Learning Strategy for Trajectory Planning and Control of UAV Swarms," *AIAA SCITECH 2024 Forum*, 2024, p. 2761.