Puffin: Making Drone Simulation Workflows More Approachable Through a Browser-Based Control Panel

Tim Nguyen
University of California, Berkeley
Berkeley, California, USA

Cody Wang
University of California, Berkeley
Berkeley, California, USA

Anjana Niranjanan
University of California, Berkeley
Berkeley, California, USA

Tien Pham
University of California, Berkeley
Berkeley, California, USA

Sharini Rao
University of California, Berkeley
Berkeley, California, USA

Abstract

Drone simulation allows engineering teams to test autonomous behavior before risking physical hardware. However, successfully running a drone simulation may require specialized knowledge of necessary dependencies: PX4, Gazebo, and ROS2. After our needfinding process with autonomous-drone club members and experienced pilots, we found that this program scatter creates mundane onboarding workflows, delays testing, and can confuse all those involved.

In order to fight these problems, We built Puffin, a browser-based dashboard that consolidates simulation startup, system status, vehicle controls, telemetry, ROS information, process monitoring, and an interactive terminal. This report documents our design process through prototyping, critique, implementation, and a five-participant usability evaluation.

Four of five evaluation participants completed our user task of taking off the drone, starting from launch of our program. Successful takeoff times were 2:35, 4:52, 3:18, and 3:42, producing an empirical completion rate of 80%. Participants required between one and four facilitator interventions and identified anywhere between three and five critical missing features. Our users valued consolidated information and detailed ROS visibility, especially on our dashboard, but largely failed with technical bits including simulation readiness and setup, details of drone control, error recovery, and inconsistent abstraction levels. These findings motivate a guided pre-flight sequence, centralized system-health feedback, plain-language ROS support, improved recovery actions, and more precise controls.

Keywords

drone simulation, human-computer interaction, usability, dashboard, PX4, ROS 2, Gazebo, onboarding

Figure 1: Puffin's visual identity

1 Introduction

Simulation is essential to autonomous-drone development because it allows teams to test behavior and inspect failures without risking physical hardware. However, the workflow itself can be difficult to access because users may need to configure several repositories, start Docker-based services, work through command-line instructions, and connect to a separate graphical environment before seeing a drone in simulation. This multi-step process is confusing and time-consuming, which is the main motivation for our project.

The steps pertaining to drone simulation belong to different technical layers, and failures that users may encounter are difficult to interpret without understanding the complete simulation stack. A user who cannot see the drone may not know where the failure might originate from. Programming leads repeatedly assist with setup, while mechanical and aerospace teammates may understand the design question they want to test but remain dependent on a programmer to execute it. Simulation becomes a specialist-controlled resource rather than a shared environment for iteration.

Puffin addresses this accessibility gap through a browser-based dashboard placed in front of the existing simulation stack. It does not replace PX4, Gazebo, ROS 2, Docker, or the underlying flight behavior but instead provides one interface for simulation startup, process health, visualization, telemetry, vehicle controls, ROS information, and terminal access.

This article contributes (1) a synthesis of need-finding evidence from users with different relationships to drones and simulation; (2) a design narrative connecting that evidence to paper prototypes, high-fidelity concepts, and the final interface; (3) quantitative and qualitative findings from a five-participant usability test; and (4) evidence-linked recommendations for future iterations.

2 Needfinding and Problem Definition

2.1 Participants

We completed four interview/observation sessions. Glen, a Berkeley UAV club programming lead, showed us his existing simulation setup and discussed on-boarding troubles he experienced with his club. Greg, the president of the club, approached drone development from an aerospace perspective, more describing testing dependencies and team workflows. Haskell, an aerial photographer and professionally licensed FPV pilot, discussed practical flight preparation and information use. Britt, an experienced FPV pilot

and drone builder, discussed learning, configuration, and simulator use. All names of interviewees are replaced by pseudonyms.

2.2 Observed Breakdowns

Glen's session showed us a workflow involving Gazebo, PX4, Docker, multiple repositories, and a VNC viewer. Because the pipeline was Linux-oriented, users on other operating systems needed additional virtualization and configuration. Essentially, the workflow lacked a clear representation of progress and readiness. For example, when the startup failed, a newcomer had to determine which component, repository, or interface was responsible. Glen reported that many of his club members struggled to get the simulator running which made on-boarding a recurring responsibility for technical leads.

Greg showed how this difficulty affected the broader team. Members who focus on the mechanical and aerospace side could formulate meaningful design questions, but still depend on programming teammates to run simulations and interpret results. Testing therefore became a handoff, slowing the cycle between design, evidence, and revision.

Haskell and Britt broadened the problem. Haskell demonstrated that users may skip relevant procedures when information is inconvenient to access. Britt described simulators that assumed substantial prior knowledge. Their experiences reinforced a general interaction principle: providing technical capability is insufficient if users cannot understand system state, information relevance, or the next action.

2.3 Creation of Puffin, Necessary Requirements

We identified three connected breakdowns. First, the simulation stack was capable but difficult to enter because setup knowledge was distributed across tools. Second, this complexity produced unequal participation: users could understand the purpose of a test, while remaining unable to execute or monitor it independently. Third, experience levels created competing interface needs. Newer users required orientation and plain-language feedback, while experienced users required logs, process state, ROS information, and granular control. These findings produced the central design opportunity: make the existing drone-simulation workflow easier to enter, observe, and act within.

We translated the research into four requirements:

(1) **Visible entry point:** Users should be able to start or access the simulation without first understanding every repository and service.
(2) **Legible system state:** The interface should communicate readiness, process health, telemetry, and available actions.
(3) **Centralized investigation:** Controls and diagnostic information should be accessible within the same environment.
(4) **Layered technical depth:** The interface should support less experienced users without removing information needed by developers.

3 Design Process

3.1 Divergent Ideation and Selection

Our first-draft concept sketches included a cross-platform launcher, simulation through SaaS, a simple guided on-boarding program, a portable Docker testing suite, natural-language control to replace CLI, collaborative simulation, adjustable complexity, a reusable behavior library, a debugging dashboard, and a mobile pre-flight application. Among those options, we chose to look into three of those options further. A consolidated dashboard directly addressed the multi-repository and dependency issue. A mobile pre-flight checklist addressed the pre-flight problem but lacked technical complexity, was too specific, and did not address our most pressing needfinding problem. A tangible paper behavior library using physical cards and QR codes lacked technical specificity as well as adaptability. We eventually landed on the dashboard approach because it addressed the broadest user group and most directly reduced the repeated simulation-access barrier, while still being easy to update and user-specific.

3.2 Primary Tasks

Our design of Puffin aims to allow users to perform the following:

(1) Configure and then launch a simulation without switching between multiple command-line tools or repositories.
(2) Monitor the drone and access relevant information without requiring extensive coding or simulation knowledge.
(3) Identify and resolve simulation problems without delegation of more experienced simulation engineers/engineering students.

3.3 Paper Prototypes

We created two paper prototypes. Prototype 1 used a dashboard-first model with visible controls and simple status updates so the system would feel easier to approach. Prototype 2 used a terminal-forward workflow with package upload and process information, more closely resembling existing robotics tools.

We tested both prototypes with three participants: Programmer, Builder, and Beginner. Programmer valued cross-platform access and technical visibility but wanted more process logging, ROS 2 interaction, and control granularity, and had a minor preference for Prototype 2. Builder valued direct controls and natural-language information and preferred Prototype 1. Beginner understood Prototype 1's controls but found the number of options overwhelming; Prototype 2's terminal was somewhat familiar, but the absence of intuitive cues, made the prototype difficult to navigate. Beginner had no strong preference.

The key finding was that accessibility and technical depth should not be treated as mutually exclusive. The next version therefore combined Prototype 1's approachable dashboard with Prototype 2's logs, process visibility, and deeper controls which led to our high-fidelity prototyping.

3.4 High-Fidelity Alternatives

Team USMNT produced three high-fidelity concepts, critiquing them using heuristics, primarily visibility of system status, recognition rather than recall, consistency, user control, and error prevention [1].

Design 1 emphasized dense service health, telemetry, mission progress, and terminal information. It offered strong visibility but created competition among top-level elements and depended heavily on raw commands. Design 2 placed Run, Pause, and Stop beside

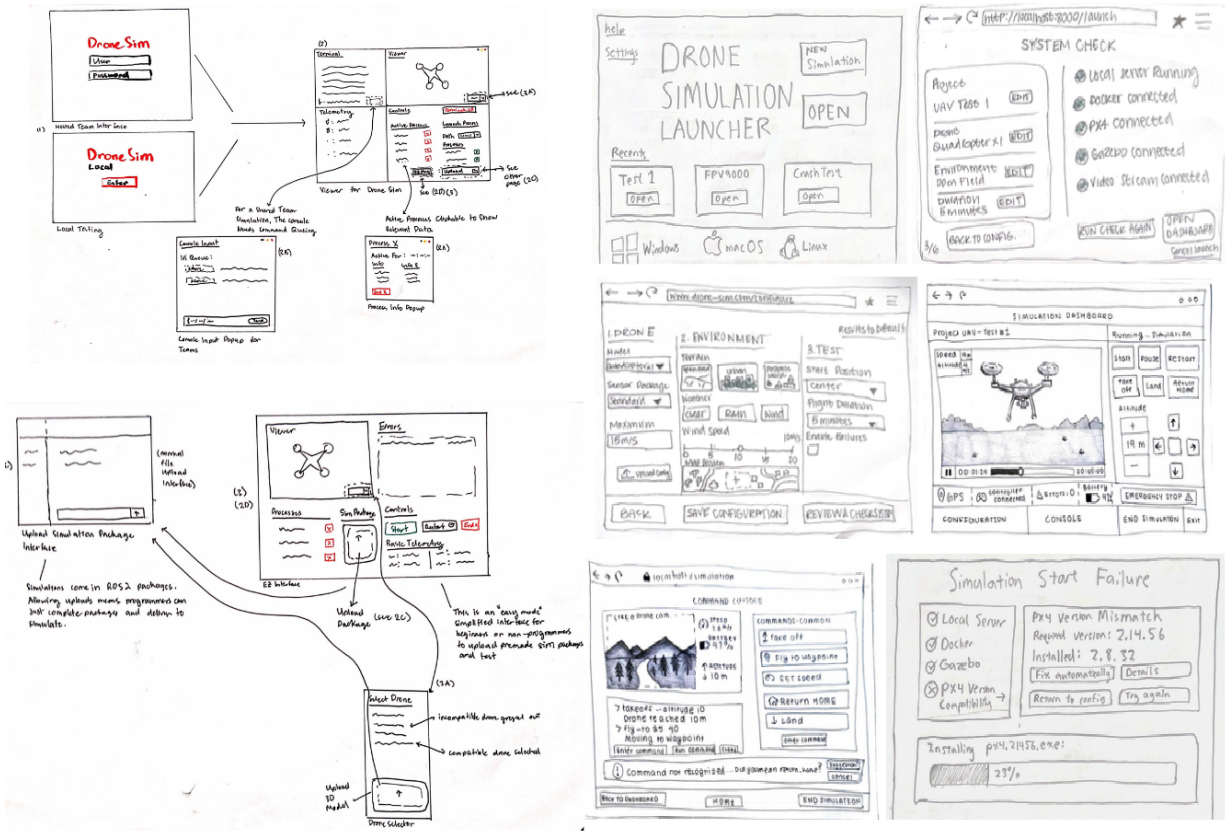

Figure 2: Dashboard- and terminal-oriented paper prototypes explored different levels of abstraction.

simulation status, improving the relationship between state and action, but the viewport remained weak and terminal control created error-prevention concerns. Design 3 used a modular light dashboard, familiar sidebar navigation, and repeated status patterns. It was the most approachable structure, although technical labels lacked explanation, whitespace reduced information density, and color carried too much status meaning.

We selected Design 3 as the foundation, then incorporated Design 2's explicit controls and Design 1's technical detail. We planned to reduce redundancy, strengthen labels, avoid color-only indicators, and make advanced information expandable.

3.5 External Critique

Two external participants evaluated the high-fidelity screens. Participant 1 was especially interested in the ROS 2 graph because it made the connection between nodes, topics, and data flow easier to see. They felt that this was helpful because that information would otherwise be hidden in terminal commands or separate tools. Participant 2 focused more on the service-health information as they liked having one place to check whether the system was running correctly when something unexpected happened. Although the two participants focused on different parts of the interface, both sessions showed that users value technical information when Puffin helps them understand how the different parts of the system relate to each other and how to proceed with our final iteration of Puffin.

4 Final Design and Implementation

Puffin allows for simulation in three stages:

First, the user enters through the launcher and checks whether required processes are active. The Start, Stop, and Reset buttons manage the environment. Second, the user monitors the simulation through the view port, telemetry, process state, flight mode, and arming state. Vehicle controls support Arm, Takeoff, Land, and Disarm when prerequisite steps have been executed. Additionally, we have introduced physical controller compatibility. Last, if behavior is unexpected or erroneous in any way, the user can easily inspect implicated processes and their health without leaving the Puffin application.

In terms of individual pages:

The dashboard combines simulation state, controls, process health, embedded visualization, PX4 telemetry, armed and flight-mode state, and access to deeper ROS information.

The ROS2 Services page shows simulation lifecycle information and controls for the offboard demonstration.

The ROS2 Graph shows active nodes, topics, and connections. The floating terminal connects through WebSocket and preserves lower-level access for experienced users.

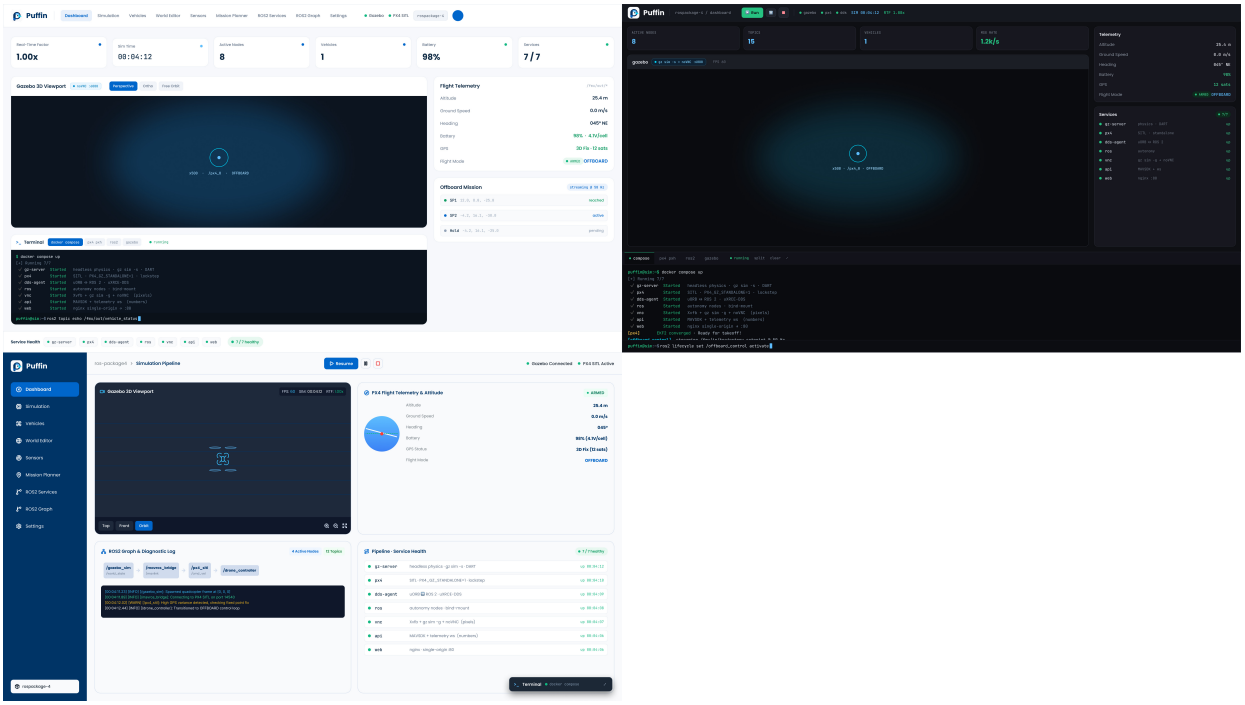

Figure 3: High-fidelity concepts explored dense monitoring, explicit simulation controls, and modular navigation.

The implemented offboard demonstration uses a ROS 2 lifecycle node to fly a 10 m square through PX4 offboard setpoints.

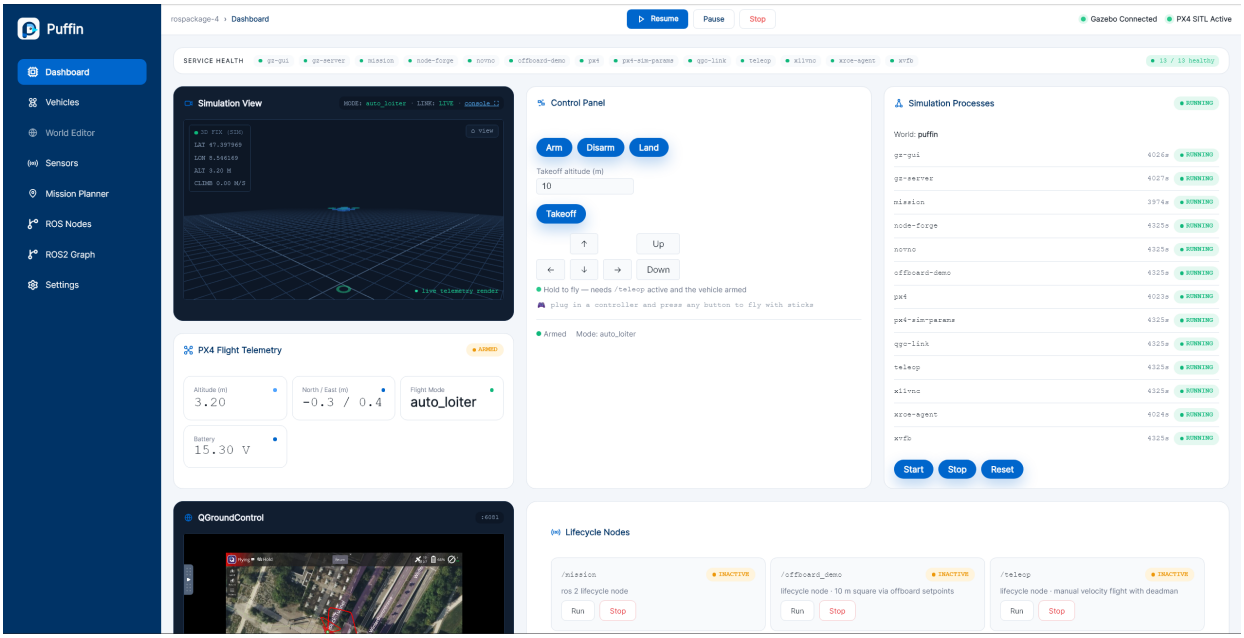

Figure 4: The final dashboard combines visualization, telemetry, process state, and vehicle controls.

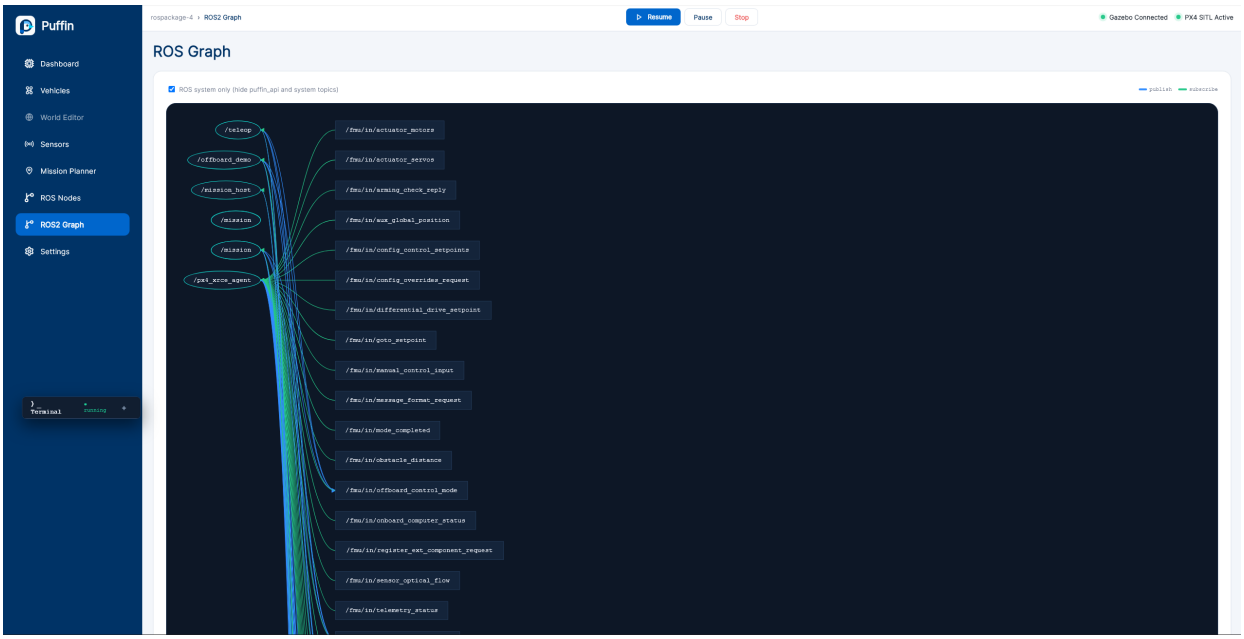

Figure 5: ROS screens preserve technical depth through visualized node relationships and lifecycle controls.

4.1 Architecture and Implementation Challenges

The browser communicates with an API exposing simulation, vehicle, ROS, authentication, settings, telemetry, and terminal functionality. HTTP requests support discrete actions, while WebSockets support telemetry and terminal interaction. Docker-based services contain the simulation dependencies.

At the M4 checkpoint and on our demo, authentication gated the simulation. This was required for submission, however in general, this application is meant to be local and used without authentication. The authentication pipeline can be used to host team-wide simulators or to host Puffin as a SaaS application. This would require containerization/sand-boxing on a per-user basis, which would be expensive without optimizations.

The largest implementation challenge was handling edge cases. Since our application is supposed to be flexible so that users can implement their own software, maintaining consistency across pages and different use cases was our biggest issue. This led to many gaps in feature completion and visual completion. For example, pause was visible but disabled; recent workspaces and workspace switching did not exist; a Sensors screen existed but was not routed by M4; and a negative takeoff altitude caused invalid behavior until validation was added.

4.2 Verification and Boundaries

The M4 pass verified launcher navigation, simulation status, Start/Stop /Reset requests, vehicle actions, the embedded viewport, terminal behavior, ROS Services, the ROS Graph, telemetry streaming, and settings. The backend had 29+ automated tests across simulation, vehicle, ROS, and authentication behavior. A manual account-flow test covered signup, login, settings modification, refresh persistence, logout, re-login, and continued persistence.

Known boundaries included desktop-oriented design without full mobile support; initial local backend setup that could take up to an hour; disabled Pause functionality; incomplete workspace selection and Sensors routing; incomplete unit and telemetry-history behavior; limited URL validation; remaining control-plane authentication work; and dependence on backend availability.

4.3 Project Availability

Prototype: http://puffin-demo.ddns.net

Repository: https://github.com/tim-nguyen0/puffin/

Demo video: https://youtu.be/nnkPGmWjSJw

5 User Evaluation

5.1 Method

Our 5 interviewees completed moderated think-aloud sessions [2]. Participants were told in advance and reminded that the interface was being evaluated rather than themselves. Participants were not given any real guidance unless they were stuck for an extended period of time with no sign of improvement.

Team USMNT's interviewers asked participants to: start the simulation and confirm readiness, monitor an autonomous square flight and land the drone, and investigate and recover from a stopped simulation. We measured time to takeoff, interviewer interventions, critical missing features identified, and takeoff completion status (boolean). Participants also completed nine five-point Likert items.

5.2 Individual Performance

Four of our five participants completed takeoff, producing a completion rate of 80%. Among successful participants, takeoff time had a reported mean of 216.8 seconds, median of 210.0 seconds, and sample standard deviation of 57.3 seconds. Interviewer interventions had a reported mean of 1.80, median of 1.00, and sample standard deviation of 1.30. Critical features missing had a reported mean and median of 4.00 and sample standard deviation of 0.71.

It is important to note that although the quantitative results show that most, 4 of 5, participants could complete the core workflow that it does not imply independent success as every participant required at least one intervention. User 1's incomplete attempt was particularly important because it exposed where a first-time mental model diverged from the intended workflow.

5.3 Likert Responses

Information relevance was generally rated more highly than its findability. This contrast is actionable; the technical information should remain available, but its hierarchy and relationship to action should and can be improved.

6 Findings and Design Implications

6.1 Readiness Was Visible but Not Decisive

Users 1, 3, and 5 paused to determine whether the simulation, connection, or required services were active. User 3 wanted a more obvious indication that launch had succeeded, in contrast to User 5 who could interpret ROS information without immediately knowing which services were healthy. Puffin displayed status, but users still had to synthesize several indicators into a readiness judgment.

Implication: Create one system-health summary connecting Gazebo, PX4 SITL, telemetry, process health, warnings, and blocking prerequisites to a clear readiness state.

6.2 Control Prerequisites Were Insufficiently Explained

User 1 couldn't locate takeoff, User 2 didn't understand that enabling teleop mode and arming were required first and asked, "Do I kill the other lifecycle first?", and User 4 wanted clearer drone orientation before issuing movement commands.

Implication: Present simulation readiness, mode selection, arming, and takeoff as an ordered pre-flight sequence. Disabled controls should explain which prerequisites remain incomplete.

6.3 Technical Information Was Useful but Fragmented

Experienced users valued the ROS Nodes and ROS Graph pages because those screens exposed information that would normally require command-line tools. However, User 1 did not understand the purpose of ROS Nodes, User 3 did not know which telemetry

Table 1: Individual performance metrics recorded in M5.

Participant	Takeoff time	Seconds	Completed	Interventions	Critical features missing
User 1	Did not complete	–	No	4	5
User 2	2:35	155	Yes	1	3
User 3	4:52	292	Yes	2	4
User 4	3:18	198	Yes	1	4
User 5	3:42	222	Yes	1	4

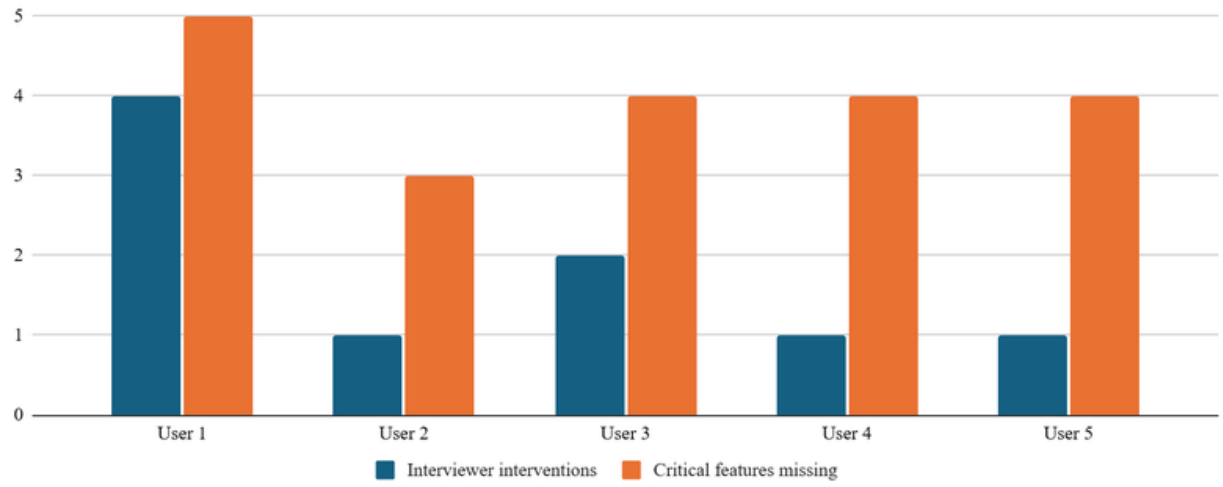

Figure 6: Recorded takeoff times, interviewer interventions, and critical missing features. User 1 did not complete takeoff.

Table 2: Five-point Likert responses from the five participants.

Item	U1	U2	U3	U4	U5
Easy to navigate and understand	1	5	2	4	3
Easy to start a simulation	2	5	3	4	4
Controls were easy and worked as expected	1	5	2	2	3
Information was relevant and useful	3	5	3	4	4
Information was easy to find	1	5	2	2	2
Simulation complexity was easier to understand than expected	1	5	2	3	3
UI was well-organized	1	5	2	3	2
Puffin would be useful in the workflow	2	4	3	3	3
Would recommend Puffin to someone with the same use case	1	4	2	3	3

values mattered, and User 4 moved repeatedly between the viewport and telemetry sections.

Implication: Preserve detailed views while adding plain-language explanations, prioritizing essential values, and grouping related spatial and status information.

6.4 Failure Detection Did Not Produce Recovery

User 1 encountered an unrecoverable viewport state after a previous session's camera had been rotated and zoomed away from the drone with no reset control available, User 3 found the volume of messages difficult to classify, and User 5 reported that warnings communicated that something happened but checked three places to understand one issue because said warnings failed to explain what to do next.

Implication: Pair warnings with affected tasks, severity, likely cause, and a recommended action. Add an in-application reset or reconnect path for recoverable failures.

6.5 Experience Level Changed What Users Needed

Newer users requested onboarding, clearer environment selection, tooltips, and simple descriptions. Experienced users expressed appreciation of the detailed information provided by the dashboard

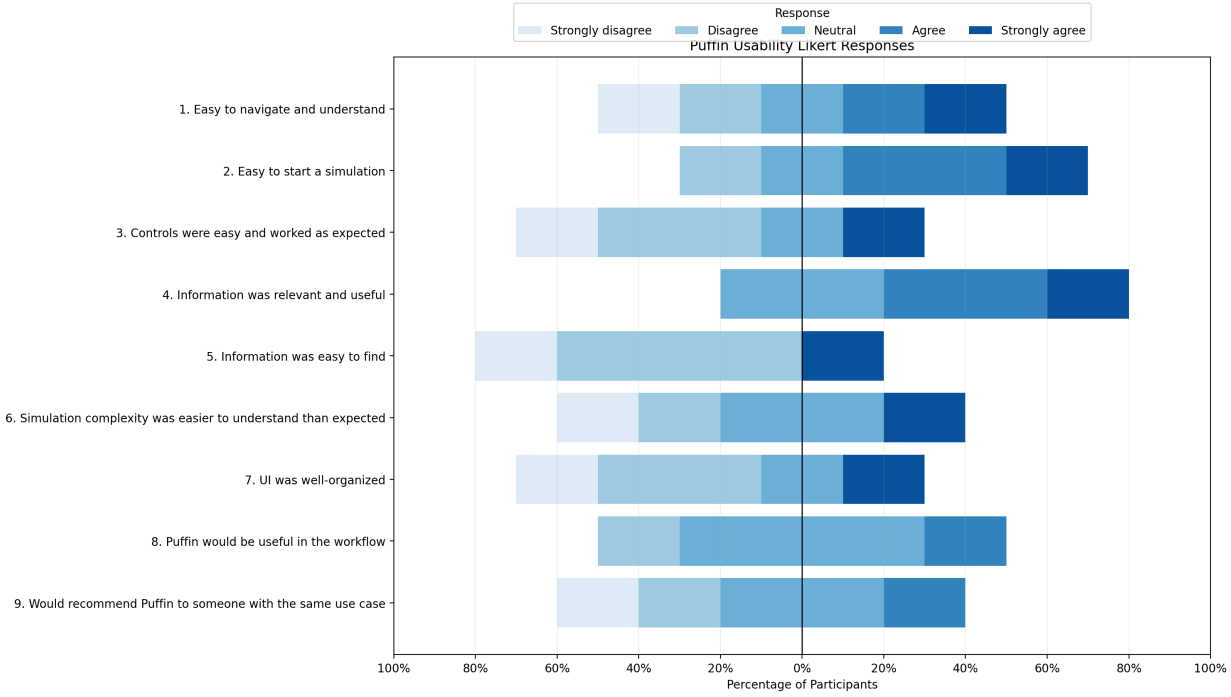

Figure 7: Responses across the nine five-point Likert items.

but requested keyboard or virtual joystick control and more configurable telemetry. This repeated the tension first observed during paper prototyping.

Implication: Use progressive disclosure. The default interface should make readiness, next action, and recovery clear, while expandable views preserve the precision experienced users need.

7 Discussion

7.1 What Puffin Accomplished

The evaluation supports Puffin's core product direction. Four of five participants completed takeoff, and participants recognized real value in having controls, simulation state, telemetry, ROS information, and troubleshooting consolidated in one application rather than scattered across tools. Notably, the detailed ROS screens were not extraneous complexity: for experienced users, they were among the most valuable features. This validates the decision to pair an approachable dashboard with deeper technical access, rather than building a purely simplified interface.

7.2 Where Puffin Fell Short

Despite this, Puffin did not fully achieve independent use for less experienced participants. User 1's incomplete takeoff and unrecoverable failure showed that exposing backend state is not equivalent to supporting recovery from it.

This points to a deeper issue: the interface reflected the system's architecture more than the user's immediate question. A user asking why takeoff is unavailable should not need to independently connect simulation state, telemetry, arming, mode, lifecycle, and warning information across several separate regions of the screen. Moreover, information was centralized at the application level, but remained fragmented at the task level.

7.3 Priorities for the Next Iteration

These findings point to eight directions for future work:

(1) Guided pre-flight sequencing for readiness, mode, arming, and takeoff.
(2) A centralized system-health panel with recommended recovery actions.
(3) Plain-language explanations for ROS states, nodes, and transitions.
(4) Reset and reconnect controls for recoverable failures.
(5) Better GPS, heading, and orientation visibility.
(6) Keyboard or virtual joystick control.
(7) A top-down map view for spatial monitoring and later geofencing.
(8) More prominent Gazebo and PX4 readiness indicators.

The first four priorities should precede broader feature expansion. They most directly address task completion and independent failure recovery which are the two gaps this evaluation identified.

8 Limitations and Future Work

The study included five participants and was designed to identify the main usability barriers rather than to support a general statistical generalization. The experience of the participants differed and the relationship between experience and success is descriptive rather than causal. The sessions were moderated, so the presence of

the facilitator may have affected persistence and confidence. Time to takeoff also combined navigation, interpretation, waiting, and control rather than isolating each sub-task.

One difficulty was not a backend failure but a residual camera state: a previous session had rotated and zoomed the viewport away from the drone, and with no camera-reset control, the participant could not recover independently. This required the facilitator to restart the simulation from the backend, which is the exact kind of failure our findings argue should instead have an in-application recovery path. The prototype was also incomplete in several areas, including Pause, workspace selection, Sensors routing, unit behavior, telemetry history, URL validation, and control-plane authentication.

Future evaluation should recruit users with a wider range of programming and drone experience and test independent first use with minimal facilitation. Each primary task should be timed separately, distinguishing interaction time from backend waiting. Recovery scenarios should include stopped services, disconnected telemetry, invalid vehicle state, incorrect lifecycle order, viewport failure, and terminal disconnection. A comparative study should test whether guided sequencing and centralized health reduce intervention without slowing experienced users.

9 Conclusion

Puffin started with the observation that drone simulation can have a high entry barrier due to a fragmented, Linux-oriented workflow. This has created steep on boarding costs for programming leads, heavy testing dependence for non-programming teammates, and uncertainty for beginner drone users.

Paper prototyping showed that beginners valued direct controls and understandable feedback from the app while experienced users required logs, process visibility, and technical depth. High-fidelity critiques led to a modular interface and combined an approachable overview with detailed ROS, telemetry, and terminal functionality.

The final evaluation demonstrated both the values and limitations of the approach. Four of five participants completed takeoff, but users still struggled to determine readiness, understand control prerequisites, recover from failure, and connect related information shown in different parts of the interface.

The main takeaway is that this dashboard works well for more experienced users, but for beginners, there are some changes that could be made in the next iterations. A true usable simulation interface needs to connect system state to meaning, meaning to action, and failure to recovery. Puffin gives a solid foundation to build on, but for beginners, the next iteration should prioritize guided pre flight sequencing, centralized system health, plain-language technical support, and recoverable failures before we expand scope further.

Acknowledgments

We would like to thank the participants who contributed to the need-finding sessions, paper-prototype testing, critique sessions, and final usability evaluation. The need-finding participants are represented through pseudonyms, and evaluation participants are identified only by participant number.

References

[1] Jakob Nielsen. 1994. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 152–158.

[2] K. Anders Ericsson and Herbert A. Simon. 1993. *Protocol Analysis: Verbal Reports as Data*. MIT Press, Cambridge, MA.