УДК 004.94:629.7
Oleh Savchenko[1], Svitlana Voitenko[2]
[1] student of group SC-615m NU "Zaporizhzhia Polytechnic"
[2] senior teacher NU "Zaporizhzhia Polytechnic"

DESIGN AND SIMULATION OF MODULAR UAV SYSTEMS USING THE KERBOT DRONE PLATFORM

O.I. Savchenko, S.V. Voitenko

Modularity plays an important role in the development of modern unmanned aerial vehicles because UAV components do not evolve at the same pace. Airframes, propulsion units, avionics, batteries, payload modules, and control software are often updated separately and later combined for a specific mission. Because of this, UAV development increasingly requires tools that support not only design but also compatibility analysis and simulation within a unified workflow.

Kerbot Drone can be considered an example of this approach. The platform is intended for modular UAV electronics design and analysis in a desktop environment. It combines schematic creation, compatibility checking, firmware-oriented context generation, analytical flight models, telemetry processing, and a custom simulation runtime called Kengine. At this stage, Kerbot Drone should be regarded not as a fully validated research simulator comparable to ROS/Gazebo-based environments, but as an integrated engineering platform for modular UAV design and simulation-oriented validation. Its practical value is that it links component configuration, preliminary firmware preparation, and runtime-oriented analysis within a single software environment.

1. Introduction

The concept of modular UAVs is well established in the scientific literature. Kotarski et al. describe modular multirotor design as a decomposition of the aircraft into major subsystems, enabling rapid replacement, maintainability, and educational experimentation through computer-aided design and rapid prototyping. Qi et al. show that the same modular principle is also important for simulation, where layered software architecture helps connect mission planning with 3D validation. Morais et al. also report that UAV simulation platforms work more effectively when they use a modular structure, support reproducible experiments, and remain closely connected with control-system development.

Kerbot Drone follows the same broad engineering direction, but from the perspective of a desktop product platform. It is a Windows-first application intended for UAV electronics design and analysis. Its current implementation includes a graphical scheme editor, project persistence mechanisms, component compatibility rules, firmware workbench logic, analytical flight and propulsion models, telemetry export, and a separate C-based runtime called Kengine. For this reason, Kerbot Drone can be interpreted as a modular UAV systems workspace in which design, verification, and simulation are connected through shared project data.

2. Modular Design Representation in Kerbot Drone

In Kerbot Drone, modularity begins at the project-model level. A UAV is represented as a structured scheme composed of components, typed connections, layout metadata, grouping information, and, when needed, a firmware workspace. This is significant because modularity is not reduced to a visual representation only; instead, it is encoded as a machine-readable engineering state. Components such as flight controllers, ESCs, motors, batteries, GPS modules, receivers, cameras, VTX modules, sensors, LEDs, buzzers, and servos can be placed and interconnected as subsystems of a larger vehicle.

This approach is consistent with the subsystem decomposition described in. However, in Kerbot Drone, this decomposition is implemented in software rather than through a physically reconfigurable frame. As a result, subsystem interaction can be evaluated before hardware assembly begins. The platform stores project files in encrypted form and preserves not only geometry but also connection semantics and firmware context, making the design artifact reusable across later stages of development.

Another important aspect is that the meaning of connections is explicitly represented. Power, signal, analog, video, and three-phase links are treated as distinct categories. This allows the software to reason about electrical and logical compatibility rather than merely display graphical links between blocks.

3. Practical Build Workflow

A typical workflow in Kerbot Drone begins with creating a project and defining the UAV scheme. The engineer places components and specifies their

interconnections, forming a structured representation of the vehicle at the subsystem level. At this stage, the platform already functions as a modular design environment rather than as a static diagram editor.

The next step is rule-based validation. Kerbot Drone includes a compatibility analyzer that determines whether a proposed connection is valid with respect to connection type, voltage range, current capacity, supported protocol, and subsystem-specific constraints. For example, the platform distinguishes correct flight-controller-to-ESC signal links from invalid power or mechanical links, evaluates battery-to-ESC voltage compatibility, and processes protocol-level cases such as UART, I2C, CRSF, PWM, and DShot. This functionality is important because it transforms the modular UAV model into a constrained engineering graph rather than a simple visual scheme.

After compatibility analysis, the workflow may proceed to firmware preparation. The firmware workbench identifies the controller platform from the scheme, determines whether the target belongs to AVR, STM32, ESP32, Arduino-class, or unknown categories, parses available ports, detects connected peripherals, and derives editable signal channels. Based on this context, it can generate initial source templates for environments such as Arduino Wiring, STM32 HAL C, ESP-IDF C, AVR bare-metal C, AVR assembly, or a portable C fallback. This does not replace embedded development itself, but it establishes a direct relationship between schematic structure and software structure. Such a relationship is especially valuable in modular UAV development, where hardware reconfiguration should be reflected in controller logic with minimal manual reinterpretation.

4. Simulation and Runtime Analysis

The simulation capabilities of Kerbot Drone are significant, although they require careful qualification. At its current level of development, the platform does not constitute a full software-in-the-loop or hardware-in-the-loop environment comparable to the ROS/Gazebo stack described by Morais et al., nor is it designed as a multi-UAV mission-planning simulator of the kind presented in. Instead, it is more accurately characterized as a two-level architecture combining simulation and analytical evaluation.

The first level is Python-based analytical modeling. The platform includes propulsion models for brushless motors, propellers, ESCs, and LiPo batteries, along with advanced flight-physics logic covering aerodynamics, thrust distribution, PID-based control behavior, energy estimation, and flight-time prediction. These models support engineering reasoning about mass, thrust, voltage, efficiency, and dynamic response trends.

The second level is the Kengine runtime. Kengine is a separate C codebase that includes aero, physics, collision, constraint, simulation, telemetry, UAV, world, and render subsystems. In practical terms, Kerbot Drone can launch a packaged engine binary, pass simulation parameters such as mass, thrust, battery

capacity, voltage, and charge, and then read state updates back into the analytics layer. The runtime exchange is implemented through simulation parameter and state files, allowing the desktop application and the engine to remain synchronized. This architectural solution preserves continuity between design data and runtime behavior.

From this perspective, Kerbot Drone reflects the layered logic described in: design data are created at the editor layer, validated at the compatibility layer, transformed into controller-aware context at the firmware layer, and then used by analytics and runtime execution. The main difference lies in scope. Kerbot Drone is currently focused on modular UAV electronics and subsystem validation, whereas and place greater emphasis on mission planning and formal simulator-validation pipelines.

5. Conclusion

Kerbot Drone demonstrates that the development of modular UAV systems can be organized as a continuous desktop engineering workflow. Its current contribution lies in the integration of four practical stages: subsystem-based project representation, rule-based compatibility analysis, controller-aware firmware context generation, and runtime-oriented simulation support. This architecture is consistent with the modular principles described in contemporary UAV research, particularly subsystem decomposition and layered simulation logic.

At the same time, the platform should be interpreted in a balanced way. At its current stage of development, Kerbot Drone should not yet be described as a fully validated high-fidelity research simulator equivalent to ROS/Gazebo SIL/HIL frameworks such as ProVANT. Instead, it is more appropriate to regard it as an evolving integrated engineering platform for UAV design, analysis, and simulation-oriented validation. Nevertheless, this result is meaningful. By reducing the gap between schematic construction, subsystem validation, preliminary firmware generation, and runtime-oriented execution, Kerbot Drone provides a credible foundation for further development toward broader control validation, richer simulation functionality, and more advanced mission-level modeling.