Modular Drone with Plug-and-Play Application PCB A Radar-Enabled Architecture for Multi-Application UAV Platforms

Manu Harsh T.S[1], Priya Darshini B[2], Sahana A[3], Ram Babu R[4]
[1,2,3]*Department. of Electronics and Communication Engineering, Atria Institute of Technology Bengaluru, India*
[4]*Guide by Assistant Professor, Department of Electronics and Communication Engineering, Atria Institute of Technology, Bengaluru, India*

***Abstract*—Buy almost any drone today and you are buying it for one job. Ask it to do something else and you are usually rewiring it, or building a new one. That is the starting complaint behind this paper, and the fix we landed on is fairly simple to state even if it was not simple to wire up: keep a fixed flightcritical core, and put everything mission-specific on a separate Application PCB that plugs into that core through one connector. Swap the card, not the drone. Our core is a Pixhawk running PX4 or ArduPilot, with a Raspberry Pi 4 sitting next to it doing the obstacle-sensing math the flight controller has no business being slowed down by. Range data comes from two RD61 60-GHz radar units, one bolted to each side of the frame, talking to the Pi over UART; the Pi decides what it is actually seeing and hands that off to the Pixhawk over MAVLink, alongside whatever the IMU, GPS, and barometer are already reporting. The Application PCB we built for this carries the radar wiring itself: connectors, an AMS1117-3.3 regulator, some decoupling capacitors, a few protection resistors, ESD diodes on the UART lines that are exposed at the connector, and status LEDs so you can tell what is happening without a laptop plugged in. Four AIR 2216 KV920 motors on 40 A ESCs handle propulsion, fed by a 14.8 V 4S 2200 mAh 60C LiPo pack through its own power module. The paper walks through all of that: the architecture, the connector, the power numbers, and the actual Mission Planner sequence we ran on the physical flight controller, plus how the design stacks up against fixed-purpose drones and the modular ideas already published. And we are upfront about what is still missing. Nothing here has flown yet, so endurance and how fast the radar can actually warn the flight controller of something in the way are still open questions, not settled ones.**

***Index Terms*—modular UAV architecture, plug-and-play application PCB, board-to-board connector, 60-GHz millimeterwave radar, Pixhawk flight controller, Raspberry Pi companion computer, MAVLink, UAV power management, Mission Planner**

I. INTRODUCTION

Drones have moved a long way past hobbyist photography. Surveillance, agriculture, industrial inspection, disaster response, environmental monitoring, mostly because they reach places and gather data that ground-based methods struggle with. But walk into almost any commercial platform and you will find it built around one fixed purpose. Retasking a camera drone for gas sensing is not a settings change. It is new wiring at best, a redesigned payload bay at worst, and that cost shows up again for every new mission a team decides it needs.

missions off one platform, through a plug-and-play Application PCB slot. Flight controller, companion computer, radar sensing, propulsion, power: all of that stays fixed and does not move between missions. Whatever is task-specific lives on a detachable Application PCB behind a standardized board-to-board interface, so swapping missions is a connector change, not a rewiring job. Obstacle safety does not depend on which module happens to be installed either. Two 60-GHz radar units, one per side, give the flight controller continuous range data no matter what the lighting or dust conditions look like outside.

There are five pieces to what we are contributing. The fixed-core, swappable-module architecture is the main one. Beyond that: the mezzanine connector standard that carries power, UART, and a module ID line together; the dual-radar sensing approach built around the RD-61; a power budget worked out for the 4S pack and 40 A ESCs actually used here; and

finally, a structural comparison that puts this design next to fixed-purpose drones and the modular concepts other groups have already published.

II. PROBLEM STATEMENT

None of the individual pieces of a general-purpose drone are hard to build on their own. What is hard is keeping flight critical reliability intact while the task-specific hardware underneath it keeps changing. Three problems in particular shaped this design.

A. Fixed, Single-Purpose Hardware

A drone wired for one job does not repurpose easily. Moving it to a different task usually means redesigning the wiring, and more often than not, part of the airframe too. That cost repeats for every new use case instead of being paid once.

B. Unreliable Ad-Hoc Wiring for Add-On Modules

Loose wiring between a main board and whatever sensor got bolted on afterward does not hold up well to vibration. Worse, there is no standardized way for the main controller to even know what, if anything, is connected.

C. Inconsistent Obstacle Awareness Across Conditions

Cameras lose reliability exactly when it matters most: low light, fog, dust, the conditions where a collision is most likely.

That pushed the design toward a ranging method that does not care about ambient light at all.

Put together, these three problems point the same direction. Fix the flight-critical hardware once, give task specific hardware a standardized and vibration-tolerant connector that can identify itself, and handle obstacle detection with a sensing technology that does not degrade with the weather.

III. OBJECTIVES

The goals below follow directly from the three problems above, and stay narrow on purpose rather than listing every feature a general-purpose drone could eventually have. Two of them are architectural: building a plug-and-play UAV where application PCBs swap without touching the flight-critical core, and keeping flight-control functions on the Pixhawk fully separate from mission-specific processing on the Application PCB. A standardized board-to-board interface carries that separation through in hardware, with a module identification line so the core always knows what is connected, and reliable UART communication ties the RD-61 radar units to the Raspberry Pi 4 while MAVLink links the Pi back to the Pixhawk.

On the sensing side, the goal is dual RD-61 60-GHz radar integration for real-time, lighting-independent obstacle detection, with detections fed into the Pixhawk's existing avoidance and stabilization loop so flight-path adjustment happens automatically rather than through a separate control path. A power budget sized around the 4S LiPo pack and the 40 A ESCs needs to cover the flight core, the radar, and the companion computer together, and the whole configuration should reach a calibrated, armable state through Mission Planner that can serve as the bench-verified starting point for flight testing later on.

IV. RELATED WORK

A. Modular and Reconfigurable UAV Architectures

Splitting a fixed flight core from swappable mission hardware is not a new idea. Baca et al. describe a family of modular UAV hardware platforms built around interchangeable frames and a shared avionics core, meant to support different payloads without redesigning the airframe each time [1]. That work nails down the mechanical mount, but leaves the electrical side loose: how the core actually talks to whatever gets plugged in, and how it knows what that thing even is. That gap is exactly what the mezzanine connector in Section IV is built to close.

B. Open-Source Autopilots as an Integration Backbone

PX4 and ArduPilot dominate research and light commercial UAV work for a simple reason. Both expose a documented, modular software stack instead of a closed one. Meier et al. describe PX4's node-based, multithreaded architecture, built so new hardware and payload modules integrate without touching the core flight-control loop [3]. Aliane's 2024 survey compares PX4, ArduPilot, and a handful of smaller projects, and lands on the conclusion that a stable, documented middleware layer, more than any particular sensor or airframe, is what lets a single autopilot stack cover fixed-wing, multirotor, and VTOL platforms alike [2]. This design leans on that

directly. MAVLink and the companion computer abstractions are used exactly as PX4/ArduPilot provide them, and the actual new contribution sits one layer below, in the standardized, self-identifying hardware connector.

C. Radar-Based Obstacle Detection for UAVs

Camera-based obstacle avoidance works fine in good light but is well known to struggle once optical contrast drops: dust, smoke, fog, glare. Millimeter-wave radar sidesteps that problem entirely, since it measures range from reflected electromagnetic waves rather than inferring it from an image. Yu et al. combine millimeter-wave radar with a monocular camera for UAV obstacle avoidance and find the radar channel holds up exactly where the camera channel gets weakest, fusing both into a 3-D obstacle estimate with an extended Kalman filter [5]. This design follows the same underlying logic in a plainer form. No fusion, just two RD-61 60-GHz units facing left and right, trading full-sphere coverage for a lighter pipeline that a companion computer with modest compute headroom can actually keep up with.

D. Summary and Positioning

Table I lines this design up against fixed-purpose drones and the modular concepts above, across four dimensions from Section II. It comes out Full on multi-application support and connector standardization, and only Partial on obstacle detection robustness, since two side-facing radar units cover lateral threats well but not the full sphere around the airframe. That trade is discussed again in Section XI.

TABLE I. Summary of Related Approach Categories

Category	Multi-App	Obst. Detect.	Std. Conn.	Low Cost
Fixed-purpose drone	None	Partial	None	Full
Generic modular (lit.)	Full	Partial	Partial	Partial
Proposed system	Full	Partial	Full	Partial

V. SYSTEM ARCHITECTURE AND HARDWARE MODE

A. Fixed-Core, Swappable-Module Architecture

Two hardware layers make up the platform. Everything that would compromise flight safety if it failed sits in the fixed core: the S500 carbon-fiber frame, four AIR 2216 KV920 motors, four 40 A ESCs, the Pixhawk, the Raspberry Pi 4, both RD-61 radar units, the GPS module, the transmitter/receiver pair, the power module, and the 4S LiPo pack. Power and data both follow one chain through the core: Battery to Power Module to Application PCB / RD-61 Radar to Raspberry Pi 4 to Pixhawk to ESC to BLDC Motors, with GPS and telemetry running alongside the Pixhawk on their own.

Whatever the mission calls for lives on the Application PCB, connected through the mezzanine interface covered in Section IV-C. As built here, that PCB hosts the two RD-61 radar interfaces: connectors, an AMS1117-3.3 linear regulator, decoupling capacitors, series resistors for LED current-limiting and UART protection, and ESD diodes on the exposed lines. Nothing about the slot itself is radar-specific though. Any module built to the same electrical interface fits the same connector.

B. Hardware Components

TABLE II. Core Hardware Components

Component	Part / Model — Specification
Frame	S500 quadcopter frame, carbon-fiber, 500 mm motor-to-motor
Flight Controller	Pixhawk, PX4
Companion Computer	Raspberry Pi 4
Radar (×2, left/right)	RD-61
Motors	AIR 2216 KV920 brushless DC, ×4
Propellers	Matched to AIR 2216 KV920 / S500 frame, ×4
ESCs	40 A, ×4
Battery	GenX LiPo, 14.8 V, 4S, 2,200 mAh, 60C
GPS Module	Standard UAV GPS/compass module, Pixhawk-compatible
Transmitter/ Receiver	RC transmitter/receiver pair for manual pilot control
Application PCB Regulator	AMS1117-3.3, 3.3 V linear regulator for radar/logic supply

Two RD-61 units are used rather than one because it doubles the sensing footprint. The left side reports what is approaching from the left, the right side from the right, and the Raspberry Pi merges both before deciding whether to flag anything to the Pixhawk. Worth flagging on its own: the AMS1117-3.3 only powers the radar and logic-level interface. It does not power the Raspberry Pi 4, which needs more current at 5 V than a linear 3.3 V regulator is meant to supply. The Pi draws instead from the power module's own dedicated 5 V output, sized for its peak load.

C. Connector and Communication Interface

Five signal classes cross the mezzanine connector between core and Application PCB: a regulated power rail, common ground, UART transmit, UART receive, and a dedicated identification line. The identification line ties through a fixed resistor value on the Application PCB side, read by the core before UART communication even opens, so the system knows what is plugged in before it tries to talk to it. ESD diodes sit on the UART lines right at the connector boundary, since those are the lines most exposed whenever a module gets unplugged and swapped in the field.

UART carries the radar-to-Raspberry-Pi link. MAVLink carries the Raspberry-Pi-to-Pixhawk link. UART won out over SPI or I²C for the radar side mainly because the RD-61 already outputs serial range data directly, without I²C's shared bus addressing overhead or SPI's tighter clock synchronization demands. MAVLink was the obvious choice for the Pi-to-Pixhawk link simply because PX4 and ArduPilot already expose it for companion-computer integration. No custom protocol needed on either end.

VI. POWER MANAGEMENT

The GenX 4S pack sits at a nominal 14.8 V, up to 16.8 V fresh off the charger, four cells in series. Its 2,200 mAh rating means it stores 2.2 Ah of charge, and the 60C discharge rating works out to roughly 132 A of sustained current (2.2 A·h × 60), comfortably above what four AIR 2216 motors pull under normal flight loads. The power module draws this bus directly for the four ESC/motor pairs and separately regulates a 5 V output for the Pixhawk, the Raspberry Pi 4, the GPS module, and, through the mezzanine connector, whatever the Application PCB needs at higher current. The AMS1117-3.3 then steps that 5 V feed down to 3.3 V locally on the Application PCB itself.

End to end, the power path looks like this. Battery (14.8 V, 4S, 2,200 mAh, 60C) feeds the Power Module, which feeds the ESCs and BLDC Motors on one branch, and in parallel feeds the Pixhawk, Raspberry Pi 4, RD-61 radar via the Application PCB, and GPS. The 40 A ESC rating leaves real margin over the AIR 2216 KV920's typical draw at this cell count and propeller size, so the ESCs are not operating anywhere near their ceiling during ordinary hover or maneuvering.

No flight-measured current or endurance number appears here. Section XI is direct about that gap. What the component ratings alone support is that the 60C battery leaves real headroom over the combined ESC draw, and that the 5 V rail feeding the Pi, Pixhawk, and radar stays electrically separate from the high-current motor rail, so switching noise on the motor side should not bleed into the companion computer or the radar's UART output.

VII. CONFIGURATION AND SYSTEM WORKFLOW

The Pixhawk gets configured and calibrated through Mission Planner before any flight is attempted; Q Ground Control works as an equivalent ground-station front end for teams standardizing on PX4 instead of ArduCopter, though Mission Planner is what was actually used here. Procedure 1 is the sequence we actually ran on the flight controller. It gets the vehicle to a calibrated, armable state, which is a different thing from flight data, and the two are kept separate throughout this paper.

Procedure 1, Mission Planner Configuration Sequence:

1) Firmware Install. PX4 or ArduCopter firmware goes onto the Pixhawk through Mission Planner's Install Firmware screen.

2) Frame Class and Type. The airframe is set as a quadcopter, 'X' frame type, matching the S500's four-arm layout, so each output channel maps to the right motor position.

3) Accelerometer Calibration. The IMU's accelerometer is calibrated across its reference

orientations so the Pixhawk can compute an accurate gravity-referenced attitude estimate.

4) Compass Calibration. The compass is calibrated separately to correct for airframe magnetic interference that the barometer and IMU alone cannot account for.

5) Radio Calibration. The transmitter/receiver pair is calibrated so Mission Planner logs the right pulse-width range for roll, pitch, throttle, yaw, and the auxiliary channels used for mode switching.

6) ESC Calibration. All four ESCs get calibrated together so each one maps the same throttle input to the same output, keeping response consistent across all four arms.

7) Flight Modes and Failsafe. Flight modes are assigned to the mode switch, and failsafe behavior for radio loss and low battery is set up before any arming is attempted.

8) Ground Verification. Mission Planner's HUD is checked for EKF health, vibration levels, and GPS lock while the vehicle stays DISARMED. Arming waits until GPS lock and EKF health both come back clean for any mission that needs them.

Once configuration is done and the vehicle is armed, obstacle sensing runs continuously alongside stabilization. Each RD-61 streams range data over UART to the Raspberry

Pi, which runs its detection logic and forwards anything flagged to the Pixhawk over MAVLink, feeding straight into the same avoidance and stabilization loop already consuming IMU, barometer, compass, and GPS data.

VIII. FUNCTIONAL MODULE INVENTORY

It helps to group the hardware into five subsystems rather than list every part flat. Flight and Propulsion is the biggest group by far: Pixhawk, S500 frame, the four AIR 2216 KV920 motors, four 40 A ESCs, and the GenX 4S battery. Obstacle Detection is just the two RD-61 units plus the Raspberry Pi's fusion logic sitting behind them. Power Management is the power module, the AMS1117-3.3, and the ESCs' current handling. Then there is the Application PCB Module itself, the radar connectors, the protection parts, and the mezzanine interface, which is the one piece of this whole list actually meant to change as new mission modules get designed. Navigation and Telemetry closes it out with the GPS module and the transmitter/receiver pair.

IX. SOFTWARE DESIGN

A. Flight Firmware and MAVLink Integration

Both PX4 and ArduCopter expose a companion-computer interface over MAVLink specifically so an external processor, the Raspberry Pi 4 here, can report sensor data or send highlevel commands without touching the flight-control loop directly. That separation is deliberate. The Pi's obstacledetection code runs as a MAVLink client rather than something with direct access to motor outputs, so a bug or crash on the radar-processing side cannot reach the motors.

B. Radar Data Processing

Each RD-61 streams its range measurements over its own UART port to the Raspberry Pi. The Pi's detection logic is written in C/C++ for the UART-facing part that has to keep up with the radar's data rate, with Python used for any ofline analysis or quick prototyping of the filtering approach before it gets ported over. The program reads both channels, applies whatever filtering the detection logic calls for, and reports one consolidated obstacle state to the Pixhawk over MAVLink. Keeping this off the Pixhawk's own microcontroller is intentional. The flight-control loop runs at a fixed, hard realtime rate that radar processing should not be allowed to jitter, and the Pi has the spare compute the radar channel actually needs.

C. PCB Design

The Application PCB was laid out in KiCad: schematic capture, placement, and routing for the radar connectors, the AMS1117-3.3 regulator circuit, decoupling capacitors, protection resistors, ESD diodes, and the status LEDs described in Section IV.

X. COMPARATIVE ANALYSIS

Table III scores all three design categories across seven dimensions tied to the problems in Section II, using the same scale as Table I: None = 0, Partial = 1, Full = 2, so 14 is the maximum.

TABLE III. Structural Feature Comparison

Dimension	FixedPurp.	Generic Mod.	Proposed

Multi-application support	None (0)	Full (2)	Full (2)
Standardized module connector	None (0)	Partial (1)	Full (2)
Obstacle detection robustness	Partial (1)	Partial (1)	Partial (1)
Lighting/weather independence	None (0)	Partial (1)	Full (2)
Maintenance / upgrade ease	None (0)	Partial (1)	Full (2)
Cost efficiency across apps	None (0)	Partial (1)	Partial (1)
Vibration-tolerant wiring	Partial (1)	Partial (1)	Full (2)
Aggregate (max 14)	2	8	12

This design lands at 12 of 14, ahead on connector standardization, lighting/weather independence, and vibration-tolerant wiring. It does not get a Full on obstacle detection robustness, and should not. Two side-facing radar units cover left and right well but leave the front, rear, and vertical approaches unwatched, so Partial is the honest score here rather than a rounded-up one. Fixed-purpose drones sit at the bottom (2), mostly reflecting almost no built-in obstacle sensing and no standardized module interface at all. None of this is a flight-tested benchmark. It communicates design scope, and Section XI says outright what it would take to turn it into one.

XI. APPLICATIONS

Pretty much any mission that reduces to “put a sensor PCB behind this connector” works here. Swap in a camera or thermal module and you have got a surveillance platform. A multispectral module turns the same drone into a crop inspection tool. Buildings, towers, pipelines: a high-zoom optical module covers industrial inspection without touching the flight core at all. Gas sensors or thermal imaging suit environmental monitoring and disaster response about equally well, and both reuse the exact same power budget and radar sensing described earlier. Even academic labs benefit, mostly because trying out a new sensor no longer means building a new airframe just to test it, thanks to the UART/MAVLink interface already being standardized.

XII. DISCUSSION AND LIMITATIONS

Sections IV through VIII address the three problems from Section II by design, but a fair account has to say directly where the platform's design still comes up short instead of glossing over it.

The biggest gap: no flight-validated data exists yet. Every number in Sections V and IX is a structural estimate or a component-datasheet value, nothing measured from a flown vehicle. Section VI's Mission Planner sequence has actually been run on the bench. The vehicle calibrates cleanly and reaches an armable DISARMED state with EKF and vibration readouts nominal. But there is been no GPS lock, no hover, no flight test logged yet, so hover current, real endurance, and radar-to-avoidance latency are all still open questions.

Obstacle coverage is lateral only, and that is a real gap rather than a rounding choice on Table III. The two RD-61 units face left and right. Nothing currently watches forward, rear, or up and down.

Latency through the radar-to-flight-controller path is also unmeasured. Obstacle detection runs on the Raspberry Pi and reaches the Pixhawk over MAVLink rather than being computed on the flight controller directly, which adds a hop whose end-to-end delay has not yet been checked against how fast the drone might be closing on an obstacle.

Connector durability under sustained vibration is assumed rather than measured. The mezzanine connector was picked specifically for standardized, vibration-tolerant mounting, but this particular connector-PCB assembly has not gone through its own vibration or cycling test on this airframe yet.

And cost efficiency is only partial, not full. Dual radar modules plus a companion computer add bill-of-materials cost over a bare-minimum fixed-purpose payload. That is a trade made on purpose for flexibility, not an oversight, but it is still a real cost a purely single-purpose drone would not carry.

XIII. FUTURE SCOPE

The next steps are not complicated to list, just time-consuming to do. Top of the list: get a GPS lock and actually fly the thing, even just a tethered hover, so

Section VI's bench numbers turn into real flight data with measured hover current and endurance we can check against the Section V budget. After that, forward-facing radar needs to happen, since the lateral-only gap from Section XI is a real blind spot and not a minor one. The MAVLink path between the Pi and the Pixhawk needs its latency actually benchmarked rather than assumed, and that number has to be checked against how fast this thing can realistically close on an obstacle before it means anything. The mezzanine connector should go through a
proper vibration and cycling test instead of relying on its rated spec alone. And down the road, the identification line deserves to grow into something closer to a capability descriptor, so a new Application PCB gets recognized on its own instead of needing a firmware change every time someone builds one.

XIV. CONCLUSION

Pulling this together: a fixed core, Pixhawk, Raspberry Pi 4, dual RD-61 radar, and a 4S LiPo chain, communicates with whatever Application PCB is plugged in through one mezzanine connector, and that single idea is what actually gets the cost and flexibility problems from Section II addressed rather than just described. The two side-facing radar units keep obstacle sensing working in lighting conditions that would give a camera-based system trouble, at the acknowledged cost of only watching the sides rather than everywhere at once. Everything in Section VI was actually run on a physical Pixhawk sitting on a bench, which is more than a paper design gets you, even though nothing has flown yet. Held up against fixed-purpose drones and the modular work surveyed in Section III, this design comes out ahead on connector standardization and lighting-independent sensing, and behind on parts of cost and full obstacle coverage, a trade-off stated here rather than hidden. The gaps in Section XI, no flight data, radar that only looks sideways, unmeasured latency, are not loose ends left behind. They form the basis for Section XII's roadmap.

ACKNOWLEDGMENT

The authors thank the Department of Electronics and Communication Engineering, Atria Institute of Technology, and their project guide, Ram Babu R, Assistant Professor, for technical guidance, laboratory access, and resources supporting this work.

REFERENCES

[1] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles," Journal of Intelligent & Robotic Systems, and arXiv:2302.04778, 2023.

[2] N. Aliane, "A survey of open-source UAV autopilots," Electronics, vol. 13, no. 23, p. 4785, 2024.

[3] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in Proc. IEEE Int. Conf. Robotics and Automation (ICRA), Seattle, WA, USA, 2015, pp. 6235–6240.

[4] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in Proc. IEEE Int. Conf. Robotics and Automation (ICRA), Barcelona, Spain, 2004, pp. 4393–4398.

[5] H. Yu, F. Zhang, P. Huang, C. Wang, and Y. Li, "Autonomous obstacle avoidance for UAV based on fusion of radar and monocular camera," in Proc. 2020 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS), 2020, pp. 5954–5961.

[6] R. Austin, Unmanned Aircraft Systems: UAV Design, Development and Deployment, 1st ed. Chichester, UK: John Wiley & Sons, 2010.

[7] R. W. Beard and T. W. McLain, Small Unmanned Aircraft: Theory and Practice. Princeton, NJ, USA: Princeton Univ. Press, 2012.

[8] M. I. Skolnik, Introduction to Radar Systems, 3rd ed. New York, NY, USA: McGraw-Hill, 2001.

[9] PX4 Development Team, "PX4 Autopilot User Guide," PX4 Open Source Project. [Online]. Available: https://docs.px4.io/main/en/

[10] ArduPilot Documentation Team, "ArduCopter Documentation," 2024. [Online]. Available: https://ardupilot.org/copter/