Autonomous Operation of Drone Using MAVLink Protocol

Madhura Pradeep Havaldar
Dept. of E&TC Engineering
Ashokrao Mane Group of Institution
Kolhapur, India
madhuhavaldar05@gmail.com

Siddhi Mahesh Sutar
Dept. of E&TC Engineering
Ashokrao Mane Group of Institution
Kolhapur, India
siddhisutar269@gmail.com

Sakshi Mahadev Patil
Dept. of E&TC Engineering
Ashokrao Mane Group of Institution
Kolhapur, India
pagoan2009@gmail.com

Dr. Sangeeta Rajendra Chougule
Dept. of E&TC Engineering
Ashokrao Mane Group of Institution
Kolhapur, India
shivsangeeta.chougule@rediffmail.com

Abstract

Drones, or Unmanned Aerial Vehicles (UAVs), are now widely used in areas like surveillance, farming, disaster relief, and goods delivery. One big problem we noticed is that most ready-made drone systems use closed, proprietary software — this makes it very hard to customize or expand them for research purposes. In this work, we built our own open-source drone system using a Raspberry Pi Zero 2 W as the brain of the drone, connected to a Pixhawk flight controller through the MAVLink protocol. The Raspberry Pi handles decision-making tasks while Pixhawk takes care of actual flying — keeping the drone stable and controlling the motors. We used freely available tools: ArduPilot for flight firmware, QGroundControl for monitoring, and PyMAVLink Python scripts for sending commands. Our tests showed the drone could take off on its own, hover at a set height, and land — all without human input. This kind of setup is low-cost and easy to modify, making it useful for students and researchers who want to experiment with drone automation.

Keywords: *Drone, UAV, Pixhawk, MAVLink Protocol, Raspberry Pi Zero 2 W, ArduPilot, Companion Computer, Open-Source Drone, PyMAVLink, QGroundControl, Autonomous Flight*

I. INTRODUCTION

Over the past few years, drones have become remarkably common — you see them being used for monitoring crops, inspecting bridges, delivering packages, and even helping during floods and earthquakes. But despite how popular they have become, most commercial drones come with locked-down software that you simply cannot change. This is a real problem for students, researchers, and small companies who want to try new ideas or add custom features to their systems.

One thing we found particularly frustrating about these closed systems is that they do not easily work with standard communication tools like MAVLink. Because of this, adding things like camera-based object detection, AI navigation, or even connecting the drone to simulation software becomes unnecessarily difficult. You either need expensive proprietary add-ons or you end up hitting a wall entirely.

Thankfully, the open-source community has stepped in with platforms like ArduPilot and PX4. These systems let you build a drone the way you want — add your own sensors, connect a small Linux computer, and program custom behavior. Tools like ROS and OpenCV can then be plugged in quite easily. This is the direction we decided to take for our project.

Today, industries need smarter drones — ones that can look at sensor data, decide what to do next, and fly without someone holding a remote controller the whole time. Closed commercial systems just cannot keep up with these demands because their architecture does not allow that kind of flexibility. Open-source drone systems, on the other hand, are designed from the ground up to be extended and improved — that is exactly what makes them so exciting for research.

The communication glue that holds everything together in our system is MAVLink — short for Micro Air Vehicle Link. It is a small, efficient messaging protocol that lets different parts of the drone talk to each other: the flight controller, the companion computer, and the ground station software all exchange data through MAVLink messages. It is lightweight enough to run on tiny embedded hardware but powerful enough to carry GPS data, battery readings, flight commands, and more in real time.

In this paper, we describe how we built such a system from scratch. Our three main contributions are: first, we connected a Raspberry Pi Zero 2 W to a Pixhawk flight controller using MAVLink over a serial connection; second, we wrote Python scripts using the PyMAVLink library to make the drone fly autonomously without any RC input; and third, we showed that this entire setup can be built cheaply using freely available hardware and software, making it accessible to students and hobbyist researchers alike.

II. LITERATURE REVIEW

A good amount of prior work has explored the idea of adding a small Linux-based computer alongside a flight controller. Koubaa et al. [1] were among the first to clearly show that pairing a single-board computer with a dedicated autopilot allows the drone to do things like image processing and autonomous navigation that the flight controller alone simply cannot handle. Their work gave us a clear model to follow.

The use of MAVLink as the go-to communication standard for drones has been well documented. Both the PX4 team [7] and a survey in MDPI Electronics [2] confirm that MAVLink has become the standard choice because it is light, fast, and works across many different hardware setups. This gave us confidence that building our system around MAVLink was the right call.

Ruiz et al. [3] took things further by showing that Linux-based boards like the Raspberry Pi and Jetson Nano can run heavy tasks like object detection and SLAM directly on the drone. This was encouraging for us because we were planning to use a Raspberry Pi as well, and it showed that even a small, low-power board could carry out meaningful computation during flight.

The ArduPilot [6] and PX4 [7] documentation also helped us understand how to properly configure MAVLink on both the companion computer and flight controller sides. These open firmware projects have made enormous contributions to UAV research by keeping everything transparent and well-documented — something we genuinely appreciated while building our system.

Security in drone communication is something we also looked into. Allouch et al. developed MAVShield, which is a lightweight way to encrypt MAVLink messages without eating up too much processing power on small embedded boards. Related work on MAVSec found that ChaCha20 encryption strikes a good balance between speed and security for these kinds of resource-limited platforms. This is something we plan to incorporate in future versions of our system.

Sarglouss and Shenouda [10] looked at how to keep the communication link between a drone and its ground station stable and low-latency, which matters a lot during live surveillance missions. Stephan [11] proposed a software layer that sits on top of MAVLink to make writing autonomous control code much simpler — this kind of abstraction is exactly what we tried to achieve with our Python scripts as well.

Finally, Patil and Pournouri [12] studied whether Linux is actually a safe choice for embedded UAV platforms. Their conclusion was reassuring — the open-source Linux community is very active in patching vulnerabilities, and the OS has a strong track record of security for embedded systems. This helped justify our decision to run Linux on the Raspberry Pi Zero 2 W.

III. PROBLEM STATEMENT

The core problem we ran into when starting this project was simple: most drones you can buy off the shelf are completely locked down. You cannot look at how the software works, you cannot add new sensors easily, and you definitely cannot reprogram the flight logic to do something creative. For a college research project with a limited budget, this is a serious barrier.

On top of that, proprietary systems often do not support MAVLink at all, which means you cannot connect them to popular tools like QGroundControl or run your own code to control the flight. Trying to add computer vision or autonomous navigation to such systems is nearly impossible without expensive proprietary SDKs. This is exactly the kind of wall that stops students and small research groups from doing meaningful work in UAV development.

Our work directly targets this problem. We designed and built a drone system using all open-source components — a Raspberry Pi running Linux, a Pixhawk flight controller, and MAVLink for communication between them. The result is a platform that anyone can replicate, modify, and build upon. It is affordable, fully transparent, and ready for the kinds of experiments that students and researchers actually want to run

.

IV. METHODOLOGY

A. Hardware Architecture

1) Pixhawk Flight Controller

The Pixhawk is an open-source flight controller that we chose as the core of our drone. It handles all the low-level stuff — reading the gyroscope, accelerometer, GPS, and barometer — and uses that data to keep the drone flying stably. Think of it as the reflexes of the drone: it reacts to changes in tilt, wind, and altitude many times per second to keep everything balanced.

What we really liked about Pixhawk is that it works with both ArduPilot and PX4 firmware, supports return-to-home, mission waypoints, and safety failsafes, and communicates over MAVLink through its serial ports. This made connecting it to our Raspberry Pi companion computer very straightforward.

Fig. 1: Pixhawk Flight Controller

2) Holybro Pixhawk 4

The Holybro Pixhawk 4 is one step up from the basic Pixhawk — it uses a faster 216 MHz processor with 512 KB RAM, includes two IMUs for better redundancy, and comes with a GPS and magnetometer already integrated. We looked at this model because it offers more interfaces (PWM, UART, CAN, I2C) which is useful when connecting multiple peripherals. It is a solid choice for research-grade builds where reliability matters.

Fig. 2: Holybro Pixhawk 4

3) Hex Cube Black

The Cube Black by Hex is designed for situations where you really cannot afford a failure mid-flight. It has three separate IMU units — if one fails, the others keep going. It also has a rubber vibration dampening mount built in, which is important when motors create strong vibrations. We came across this in our research when looking at professional survey drones; it is a popular choice for mapping applications.

Fig. 3: Hex Cube Black

4) Holybro Pixhawk 6

The Pixhawk 6 is the newest model in the family and honestly quite impressive — it runs at 480 MHz which is significantly faster than previous versions, supports a wide range of interfaces, and includes improved vibration isolation. For our project we noted it as a future upgrade path; if we wanted to run more complex onboard computations directly on the flight controller, the Pixhawk 6 would be the natural next step.

Fig. 4: Holybro Pixhawk 6

Fig. 5: Complete UAV Hardware Components used in the Proposed System

TABLE I: Comparative Analysis of Pixhawk Flight Controllers

Model	Processor / Clock	Sensors & IMU	Memory	Key Highlights
Pixhawk 1	STM32 ~168 MHz	Single IMU, GPS, Baro	256 KB RAM, 2 MB Flash	Budget-friendly, MAVLink
Pixhawk 4	STM32F765 216 MHz	Dual IMU, Baro, GPS+Mag	512 KB RAM, 2 MB Flash	Open-source standard
Cube Black	STM32 180 MHz	Triple IMU, Dual Baro	256 KB RAM, 2 MB Flash	High reliability, redundancy
Pixhawk 6	STM32H7 480 MHz	Dual IMU, Baro, GPS	1 MB SRAM, 2 MB Flash	Modern, powerful, rich I/O

B. Software Architecture

1) MAVLink Protocol

MAVLink is the language our drone speaks. Every piece of information — where the drone is, how fast the battery is draining, what flight mode it is in, what commands to execute — is packaged into small MAVLink messages and sent back and forth between the Pixhawk, our Raspberry Pi, and QGroundControl. It works over serial cables, UDP, or TCP, which makes it flexible enough to use whether you are sitting next to the drone or monitoring it from a laptop across the room.

The newer MAVLink 2 version added message signing so that only trusted sources can send flight commands — an important safety feature. In our implementation, we used two main types of interaction: reading telemetry data from the drone, and sending commands like ARM, TAKEOFF, and LAND. Each command we sent received a COMMAND_ACK reply from the Pixhawk telling us whether it worked or not, which helped a lot during debugging.

2) Raspberry Pi Zero 2 W (Companion Computer)

We picked the Raspberry Pi Zero 2 W as our companion computer mainly because of its size and price — it is tiny, costs very little, and yet has a quad-core processor that is genuinely capable. Earlier Raspberry Pi Zero models were too slow for our needs, but the Zero 2 W hit the sweet spot. It also has built-in Wi-Fi which meant we could SSH into it wirelessly during testing without needing extra hardware.

In our setup, the Pi runs the Python scripts that tell the drone what to do. It reads incoming telemetry, decides when conditions are right to move to the next step, logs data, and sends commands to the Pixhawk over the UART serial connection. The two-way communication means the Pi is not just giving orders — it is also listening to what the drone reports back.

3) ArduPilot Firmware

ArduPilot is the firmware we flashed onto the Pixhawk. It is open-source, very well maintained, and has a huge community around it. For our quadcopter we used ArduCopter, which is the multirotor variant. It gave us features like automatic stabilization, GPS-based position hold, return-to-home on low battery, and — most importantly for our project — a GUIDED mode that lets the companion computer take over and send position and velocity commands directly.

4) QGroundControl (Ground Control Station)

QGroundControl was our ground station software. Before flying, we used it to calibrate sensors, set up safety parameters, and verify that everything was connected properly. During test flights, it showed us live data — altitude, GPS lock, battery level, motor output — all on one screen. It runs on Windows, Linux, and even Android, which was convenient since we could monitor the drone from a phone. It communicates entirely through MAVLink, so it fits perfectly into our setup.

5) PyMAVLink Library

PyMAVLink is the Python library we used to write our autonomous flight scripts. With just a few lines of code, we could open a serial connection to the Pixhawk, subscribe to telemetry messages, check the drone's altitude, and send commands like arm, takeoff, and land. Writing the code was actually quite fun once we understood how MAVLink message types work. This library essentially became the core of everything autonomous in our system.

V. RESULTS AND DISCUSSION

A. System Integration and Communication

The first thing we tested was whether the Raspberry Pi and Pixhawk could actually talk to each other. After connecting them over UART and configuring the baud rate correctly, we ran a simple Python script that listened for MAVLink heartbeat messages. Within a few seconds, the heartbeat came through — that was genuinely exciting. From that point, we could see live telemetry flowing: GPS coordinates, altitude readings, battery voltage, and flight mode status all appearing on our laptop screen through QGroundControl.

B. Autonomous Flight Execution

For our main test, we wrote a Python script that would fly the drone completely on its own. We specified a target altitude of 5 meters and told it to hover there for 20 seconds before landing. Here is exactly what happened when we ran the script:

- Successful motor arming via MAV_CMD_COMPONENT_ARM_DISARM command
- Automatic transition to GUIDED flight mode
- Autonomous takeoff execution to a target altitude of 5 meters
- Stable hover maintenance for approximately 20 seconds
- Controlled autonomous landing sequence initiation
- Post-landing transition to LOITER mode confirming mission completion

Fig. 6: Autonomous UAV Flight Operation — Takeoff and Hover Sequence

Throughout the whole flight, the Raspberry Pi was logging every telemetry message it received — GPS position, altitude, battery percentage, and flight mode. After landing, we reviewed the logs and confirmed that every command we sent received a successful COMMAND_ACK response from the Pixhawk. No errors, no dropped messages. The communication channel held up perfectly for the entire mission.

C. Performance Assessment

Looking at the overall results, we are happy with how well the system performed. The drone flew autonomously without any manual input, held its altitude steadily, and landed cleanly. The division of responsibilities worked exactly as we planned — the Pixhawk handled the fast, low-level stability control while the Raspberry Pi focused on the bigger picture of mission logic. Neither component got in the way of the other.

Perhaps most importantly for other students and researchers reading this: the entire hardware cost was quite modest. The Raspberry Pi Zero 2 W is very affordable, and Pixhawk-compatible boards are available at reasonable prices. Combined with free software, this means a functional autonomous drone research platform is within reach even for groups with tight budgets.

VI. CONCLUSION AND FUTURE SCOPE

A. Conclusion

To summarize what we set out to do and what we actually achieved: we successfully built a working autonomous drone system using a Raspberry Pi Zero 2 W, a Pixhawk flight controller, and MAVLink as the communication bridge between them. The drone armed itself, took off to a target height, hovered steadily, and landed — all through commands sent by Python code running on a tiny Linux computer. That is a real result, not just a simulation.

MAVLink proved to be an excellent choice for the communication layer — it was reliable, fast enough for real-time control, and well-supported by the tools we used. Using only open-source software meant we had full access to the code at every layer of the stack, which helped enormously when debugging unexpected behaviour. And since everything is free, anyone can replicate our work without needing to spend money on licenses.

We believe this kind of setup — affordable, open, and modifiable — is exactly what the UAV research community needs more of. Our work shows that you do not need expensive commercial platforms to do serious drone research. A Raspberry Pi, a Pixhawk, and some Python code is enough to get started, and the sky is literally the limit from there.

B. Future Scope

There is a lot we want to do next with this platform. The most exciting direction is adding a camera and running OpenCV or TensorFlow Lite on the Pi to give the drone the ability to see and react to what is around it — tracking a moving object, detecting obstacles, or following a path. We also want to try SLAM so the drone can navigate indoors where GPS does not work.

Another thing we are keen to explore is flying multiple drones together — using MAVLink to coordinate a small swarm of UAVs working on a shared task. On the security side, we plan to implement MAVShield encryption in future builds to protect our MAVLink communication from potential interference or hijacking. Overall, the foundation we built here opens up many interesting paths forward, and we look forward to continuing this work.

REFERENCES

[1] A. Koubaa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, "MAVLink communication protocol in UAV systems," in Proc. Int. Conf. on Unmanned Aircraft Systems (ICUAS), 2019.

[2] L. Delvaux and D. Naro, "Autopilot and companion computer for unmanned aerial vehicle: A survey," Electronics, MDPI, vol. 12, no. 3, 2023.

[3] D. V. Ruiz, L. A. Vidal, and E. Todt, "Integration of Linux, ROS, Android, and UAV systems for onboard intelligence," in Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), 2021.

[4] A. Toma, N. Cecchinato, G. Ferrin, and I. Scagnetto, "MAVLink-based control coordination of a multi-drone cluster for intelligence, surveillance and reconnaissance tasks," IEEE Access, 2024.

[5] S. Sharma, "Drone Development: From Concept to Flight — Design, Assembly and Applications of UAV," 2nd ed. New Delhi: TechPress, 2024.

[6] E. Ebeid, M. Skriver, and J. Jin, "A survey on open-source flight control platforms of unmanned aerial vehicles," in Proc. Euromicro Conf. on Digital System Design (DSD), 2017.

[7] PX4 Development Team, "PX4 Autopilot User Guide," PX4 Documentation, 2023. [Online]. Available: https://docs.px4.io

[8] P. Burdiakowski, N. Ramjooy, V. Estrela, and J. Hemanath, "Open source software and hardware in UAV: A critical review," in Proc. Int. Conf. on Emerging Trends in Engineering (ICETE), 2020.

[9] H. M. Qays, B. A. Jumaa, and A. D. Salman, "Design and implementation of autonomous quadcopter using SITL simulation," Iraqi J. Computers, Communications, Control & Systems Engineering, vol. 20, no. 2, 2020.

[10] A. Sarglouss and A. Shenouda, "Study and development of UAV remote operation via air-link," J. Aerospace Engineering, vol. 36, no. 4, 2023.

[11] M. Stephan, "Autonomous UAV MAVLink abstraction layer for simplified communication with control applications," M.S. thesis, Dept. Comput. Sci., Technical Univ., 2020.

[12] D. Patil and S. Pournouri, "Evaluating the security of open source Linux operating systems in embedded platforms," in Proc. Int. Conf. on Cyber Security and Privacy (CSP), 2023.

[13] M. C. Dobrea, C. Santora, and F. F. Santora, "A UAV development platform for intelligent applications," in Proc. IEEE Int. Conf. on Automation, Quality and Testing, Robotics (AQTR), 2019.

[14] G. Bressan, "Hardware-software architecture, code generation and control for multirotor UAVs," Ph.D. dissertation, Dept. Electrical Eng., Univ. of Padova, 2017.

[15] A. P. Lamping, "Multi-UAV control and supervision with ROS," M.S. thesis, Dept. Aerospace Eng., Delft University of Technology, 2018.