AERPAW Vehicles: Hardware and Software Choices

Mark Funderburk[*]
mtfunder@ncsu.edu

John Kesler[*]
jckesle2@ncsu.edu

Keshav Sridhar[†]
ksridha3@ncsu.edu

Mihail L. Sichitiu[*]
mlsichit@ncsu.edu

İsmail Güvenç[*]
iguvenc@ncsu.edu

Rudra Dutta[†]
rdutta@ncsu.edu

Thomas Zajkowski[‡]
tjzajkow@ncsu.edu

Vuk Marojevic[§]
vuk.marojevic@ece.msstate.edu

ABSTRACT

Aerial Experimentation and Research Platform for Advanced Wireless (AERPAW) is an advanced wireless research platform centered around fully programmable radios and fully programmable vehicles. In this paper we detail the vehicle aspects of the testbed, including the AERPAW UAVs, UGVs, as well as the hardware and software choices made by the team, as well as our experience earned in the past few years.

1. INTRODUCTION

AERPAW is one of the four PAWR platforms. All PAWR platforms are advanced wireless research platforms, allowing researchers from academia and industry to setup experiments involving wireless communications and gather data that would be otherwise difficult or expensive to obtain. For AERPAW, one of the distinguishing features is the availability of fully programmable unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs). The AERPAW vehicles allow experimenters to precisely control the position and orientation of the radio equipment they use in their experiments, allowing for flexible, controllable, and reproducible results.

2. PORTABLE NODES

At this time, in AERPAW the computing nodes in the testbed are segregated into three different types: **Portable Radio Nodes**, which are normally carried by one of the AERPAW vehicles during an experiment (although they can also be placed at a fixed location); **Fixed Radio Nodes**, which are installed at radio towers, light poles, or rooftop locations; **Cloud Nodes**, which have no radios, but are available for experimenters' processes.

In AERPAW, the portable nodes have been developed independently from the vehicles, and they are largely interchangeable: any large vehicle can carry any of the large portable nodes (LPNs) and any of the small portable nodes (SPNs), and any of the small vehicles can carry any of the SPNs.

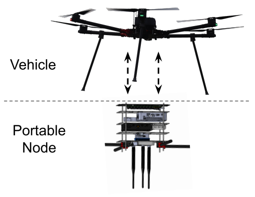

Figure 1: Portable Node Vehicle Interface

Each portable node is centered around a companion computer, with its associated power adapter, and radios it controls. The LPNs feature SDRs: B205mini, and B210, which are fully accessible by the experimenters. The SDRs have their associated front end (1W wide-band power amplifier, low

[*]Dept. of Electrical and Computer Eng., NC State University, Raleigh, NC
[†]Dept. of Computer Science, NC State University, Raleigh, NC
[‡]NextGen Air Transportation Program, NC State University, Raleigh, NC
[§]Dept. of Electrical and Computer Eng., Mississippi State University, MS 39762

DroNet'22, July 1, 2022, Portland, OR, USA
2022 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-9405-5/22/07 ... $15.00
https://doi.org/10.1145/3539493.3539583

noise amplifier, and filters) and wide-band antennas. In addition to the experimenter accessible SDR, there is a monitoring SDR that is used for capturing RF transmissions from the experimental radios and verifying compliance with FCC rules. The SPNs have a lighter and less powerful computer, that is capable of controlling off-the shelf radios (e.g., cellular modems, LoRa cards, WiFi cards), but not powerful enough to process data from an SDR.

The lightest LPN weighs three kilograms, but several new larger portable nodes are currently being developed weighing over five kilograms. The lightest SPN weighs 300g, but several versions approaching 1kg are being developed.

Since, for AERPAW, the LPNs and the SPNs are effectively the UAV and UGV payloads, the UAVs and UGV have been designed with these weights in minds: 3kg for the Large AERPAW Multicopter (LAM6) and 500g for the Small AERPAW Multicopter (SAM4).

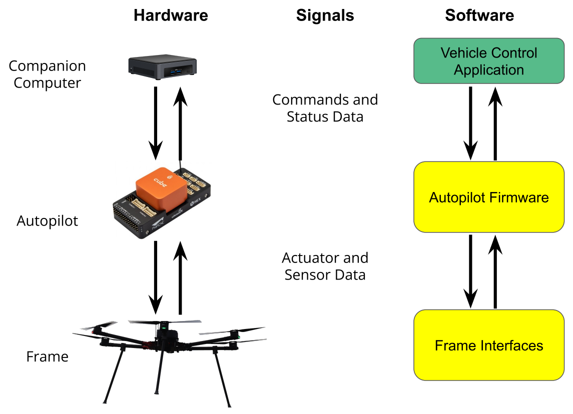

Figure 2: Simplified Vehicle System Hardware and Software Stack.

3. AERPAW VEHICLE HARDWARE

The general stack for the AERPAW vehicles is shown in Fig. 2: the experimenter's software running on the companion computer on the portable node sends commands to the autopilot, which, in turn, controls the vehicle frame, while sending status information to the experimenter's software, thus closing the loop. Status information typically offers information on the current speed, position and orientation of the vehicle, battery information, etc. Typical commands allow an experimenter to command a target position and orientation (either relative or absolute), as well as controlling the linear and angular velocities of the UAV.

3.1 UAV Options Considered

Many times when talking to our colleagues we are asked why we chose to build our own UAVs when so many other commercial solutions exist on the market. In what follows, we present the solutions we considered and how we arrived to our current design.

3.1.1 Commercial UAV Frame and Autopilot

For AERPAW there are several solutions for each of the elements shown in Fig. 2. One very popular solution for wireless researchers, is to choose a commercial UAV (including the autopilot) and put a portable node on top of this vehicle. For example, the Matrice 600 from DJI is a very capable, well trusted UAV that can fulfill a variety of roles, including wireless research [1–3]. In this case, the commercial UAV fulfills the bottom two roles on the stack in Fig. 2, and the vehicle control software, running on the companion computer will use the API provide by the commercial manufacturer. Many manufacturers (including DJI) provide an open API for accepting commands for their autopilots and providing vehicle status information.

The main drawback (in our opinion) of this solution is the lack of open access to the commercial UAV software and hardware. While this closeness is understandable from a commercial perspective, it limits what a developer can build on top of such a platform. Furthermore, if the manufacturer stops supporting the system at any time, we may have to scramble to find an alternate solution. As to underline the possibility of such an outcome, the Matrice 600 is currently no longer in production, being replaced by the Matrice 600 Pro, which uses a different autopilot. It is impossible to know if the new autopilot behaves identical to the old one. Furthermore, due to the closeness of the system, it is impractical to develop a realistic emulator for the commercial system; AERPAW is heavily relying on a realistic emulation environment for experiment development. Finally, the flexibility of the system is limited: if we need a larger battery, we cannot include it in the system; if we need a more powerful set of motors and propellers, we cannot include them in the system, etc.

3.1.2 Commercial UAV Frame and Open Autopilot

Another possibility is to completely remove the autopilot from a commercial frame and use it with an open source autopilot. This eliminates the need for building the vehicle frame, and solves the problems related to the closed design for the commercial autopilot (i.e., has an open APIs and a reliable emulator). However, this does not solve the problems regarding the long-term availability of the frame, as well as the flexibility of the frame (e.g., upgrading motors, batteries, etc.). Furthermore, the frame itself is typically not optimized for wireless research, but rather for carrying a camera and its associated gimbal, so it would require further modifications. Finally, many of the advantages of the commercial frame (e.g., the battery management system, redundant GPS) would likely be inaccessible to an alternate autopilot.

3.1.3 *Off the Shelf UAV Frame and Open Autopilot*

Yet another possibility we considered in AERPAW is to use an off-the-shelf UAV frame and fit it with a propulsion system and open source autopilot. This was, in fact, our first iteration to the AERPAW LAM. Theoretically, this is a good compromise in flexibility and effort in developing a frame. The first problem we encountered when trying to mount the motors to a commercial large octocopter frame was that the motors we selected did not fit the motor mounts that the frame came with. So, we designed new motor mounts, and cut them out of carbon fiber plate, to fit the motors to the frame. We then realized that the current top plate of the frame cannot accommodate our large batteries, so we designed a new top plate. Then the bottom plate was unfit to take the full size of our large portable nodes, so we designed and cut a new bottom plate. Then the legs of the frame gave out, so we designed new legs. By the time we were done with the frame, only the arms (which were plain carbon tubes) remained from the original custom frame. The final nail in the coffin was the fact that by the time we completed the first UAV prototype, the frame was no longer in production.

3.2 The AERPAW UAVs

As outlined in Section 3.1, after carefully considering the hardware and software options, we decided to design the AERPAW UAVs from scratch, and pair them with an open source autopilot.

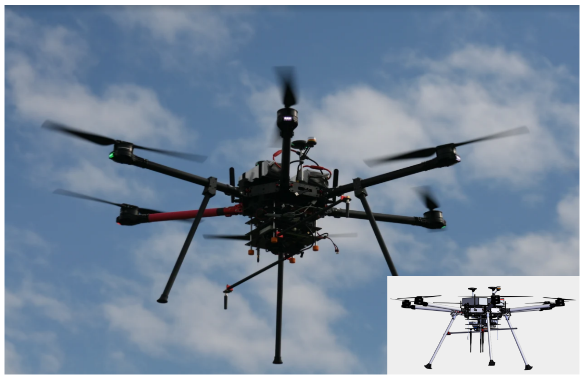

Figure 3: Large AERPAW Multicopter (LAM6) with Portable Node and CAD model.

3.2.1 *The Large AERPAW Multicopter (LAM6)*

The large AERPAW multicopter, shown in Fig. 3, has been fully modeled in a CAD system (SOLIDWORKS), and then manufactured from stock components that are readily available: the arms and legs are carbon fiber tubes, the top, bottom and the many side plates are cut from carbon fiber plates on a CNC Router, many of the joints are made from 3D printed parts with carbon fiber reinforcements. For propulsion we use the XRotor Pro X6 system with integrated electronic speed controllers (ESCs) and motors, spinning 6 x 23” propellers, each with a max thrust of 12kg, thus allowing for a safe all up weight (AUW) of up to 36kg. FAA rules, however, limit the LAM6 AUW to 25kg (55lb). In the standard configuration, the frame weighs 7kg, the batteries are 5kg (30Ah 12S1P), and the payload (a regular LPN) is 3kg. In this configuration, the current consumption is about 40A while hovering and the hovering endurance is over 40 minutes. The maximum speed (electronically limited) is 30m/s.

The LAM6 can lift up to (and over) the FAA small UAV weight limit in its current configuration. However, one of the big advantages of building our own UAVs is that, if, and when needed, we can very easily swap the arms of the LAM6 for longer ones, and the motors to larger motors, thus being able to lift much larger payloads (up to 50kg), assuming we can obtain an FAA exemption from the Part 107 rules. The frame features a simple, but effective and secure clamping system that allows for a fast swap (10-20 seconds) of the batteries and of the portable node.

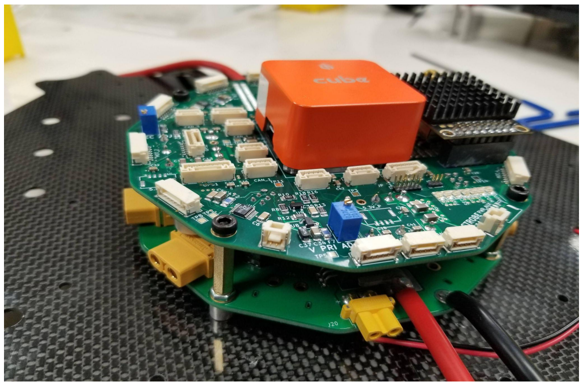

Figure 4: Two LAM6 PCBs: control on top and power distribution on the bottom.

The second significant effort in the development of the LAM6 vehicle has been the design and implementation of two custom PCBs (shown in Fig. 4). The first PCB aggregates all the control parts required by the UAV: the autopilot itself (the Cube Orange), connectors for all external components to the autopilot (2 x GPS + compass, six ESCs, USB control from the companion computer), voltage regulators for the companion computer (19V 10A), telemetry, RC receiver, extension headers for a future expansion, as well as independent status and control of power distribution at various subsystems (autopilot, motors, payload). The second PCB has the far easier (but equally critical) function of controlling (on/off) and distributing power to the six motors of the LAM6. The task is not trivial, since each motor consumes almost 60A at maximum throttle.

The LAM6 are the workhorses of the AERPAW experiments, and most of our users at this time are interested in experiments involving SDRs, and correspondingly the heavy front end, as well as the large (and heavy) companion computer required by these SDRs, requiring the large multicopter to be able to efficiently (and safely) lift them. The LAM6 is a wonderful flying machine, but it is heavy and expensive to build (a bit over \$5,000 without counting labor) and maintain. Going forward we saw the need for a smaller, lighter multicopter, with a lower cost (well under \$2,000), which led to the SAM4 (see Fig. 5).

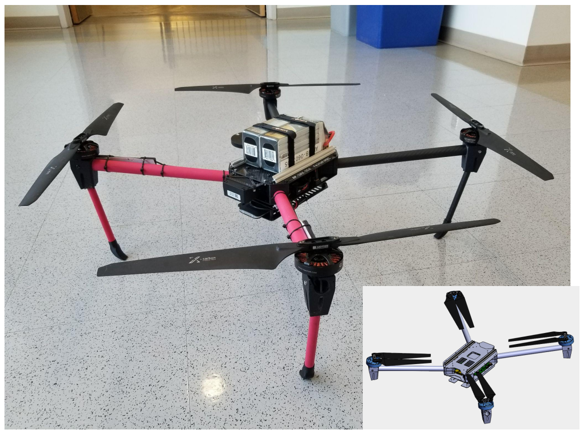
Figure 5: The Small AERPAW Multicopter (SAM4) and its CAD model.

3.2.2 *The Small AERPAW Multicopter (SAM4)*

SAM4 is a significantly smaller multicopter, with four 18" propellers, powered by a 6S2P battery pack (14Ah) in the standard configuration. For the standard small portable node (built around a LattePanda computer), under 500g, the flight time of SMA4 is similar to that of the LAM6 (a bit over 40 minutes). However, if a longer flight time is required, SAM4 can hover for *82 minutes* with a 30Ah 6S battery with a 300g small portable node. We foresee that we will employ the SAM4 for all experiments *not* requiring SDRs. If needed, SAM4 can safely lift up to 3kg of payload, albeit at a significantly reduced endurance.

Similar to the design philosophy employed for the LAM6, the SAM4 has been designed in CAD and constructed from standard components (carbon fiber tubes and plates, cut on the CNC routers and 3D printed parts as needed). A single custom PCB fulfills a similar role as the two PCBs of the LAM6 (i.e, integrated control and power distribution).

3.2.3 *The AERPAW Test Stand*

Particularly useful for the LAM6, we designed and developed a test stand that allows a UAV to exercise four

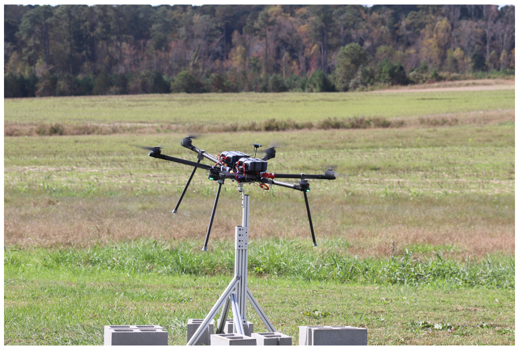
Figure 6: LAM6 on test stand.

of the total six degrees of freedom experienced in flight, while staying firmly on the ground (horizontal translation is restricted). The test stand allows for safely testing motors, calibration of the stability of the aircraft, as well as testing the components at maximum power. The test stand is securely held in place by eight concrete blocks (138 kg total).

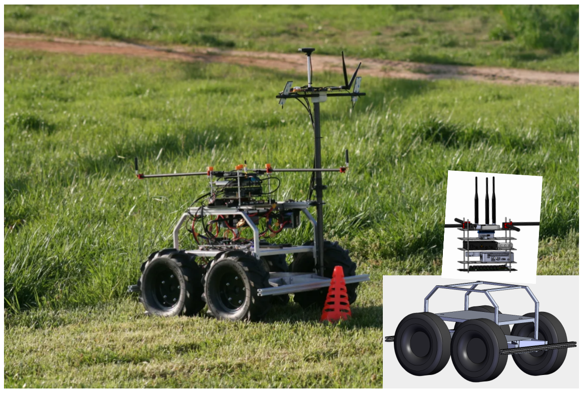
Figure 7: AERPAW Rover with a large portable node.

3.3 The AERPAW UGV

In addition to the two UAVs, AERPAW provides a simple unmanned ground vehicle (UGV) that is capable of carrying the LPNs, while also being fully controllable by the experimenter (albeit with additional restrictions on where it can travel, as our field has a few creeks crisscrossing it). Unlike for the UAVs, for the rover we chose an off-the-shelf vehicle frame (IG52-DB4 from SuperDroidRobots) and populated it with our open source autopilot (see Section 3.4) and adapters for our battery and portable nodes mounting systems.

The resulting system works very well on relatively smooth surfaces (dirt roads, mowed grass, etc.); however, on rough terrain (e.g., a plowed field), the lack of suspension on the rover results in a very rough ride.

3.4 AERPAW Autopilot

Our UAVs (LAM6 and SAM4), as well as the rover use the same off-the-shelf autopilot, the Cube Orange [4], featuring an open architecture and good support from the ArduPilot community. The autopilot is an evolution of the Pixhawk open source autopilot built around a 32bit ARM STM32H753 Cortex-M7 (with DP-FPU), and features three redundant IMUs on temperature controlled boards, two barometers, redundant power supplies with automatic fail-over, connectors for two GPS receivers, three magnetometers, and plenty of spare buses for additional sensors and auxiliary systems. The Cube Orange is well documented and there are a variety of carrier boards that can be made to work with it. In our case, the autopilot connects directly to the custom PCBs for both LAM6 and SAM4, and uses a standard carrier board for the rover.

For autopilot firmware, the two most popular autopilot options are ArduPilot [5] and PX4 [6]. We chose ArduPilot, due to its stability and due to our good experiences with the emulation environment of ArduPilot. Both ArduPilot and PX4 support MAVLink [7], an open communication protocol developed for small UAVs. The advantage of using MAVLink is that there is a large software base allowing us and the experimenters to easily develop portable vehicle control software as detailed in the next section.

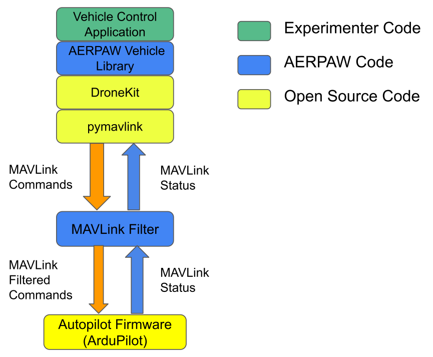

Figure 8: AERPAW Vehicle Software.

4. AERPAW VEHICLE SOFTWARE

Figure 8 shows the AERPAW vehicle control stack with the *recommended* vehicle control application framework. At the top of the stack the experimenters can either modify examples provided by the AERPAW team, or write their own vehicle control applications. The common *requirement* to any of the vehicle control applications is that they generate (and consume) MAVLink [7] messages.

4.1 MAVLink Filter

Through the AERPAW vehicle application, the experimenters can generate dangerous, or even disastrous MAVLink commands (for example, there are MAVLink commands that can stop all the drone motors mid-air, and/or override the safety pilot's commands). For this reason we developed a MAVLink filter that lets a very small subset of commands pass to the vehicle autopilot. Furthermore, even for the allowed commands, only a subset of parameters are accepted.

For UAVs, only takeoff, land, simple goto, and yaw commands are accepted at this time. For takeoff, the minimum and maximum altitudes are 20m and 100m, respectively. Landing is only permitted within 5m of the take-off location. Each goto command is checked for altitude, and for not exiting the geofenced area of our flying field. In addition, for the UGVs, we have defined no-go zones where there are creaks and other obstacles on our airfield, and each goto command is checked to not cross any of the no-go zones.

In contrast to the draconian command filtering, the MAVLink filter allows *all* the status MAVLink messages to flow from the autopilot toward the vehicle control application. This provides the vehicle control application regular updates on the position and attitude of the vehicle, as well as other information (e.g, air speed, ground speed, battery level, vehicle mode, etc.).

4.2 Vehicle Application Frameworks

The first thing that we need to emphasize is that AERPAW experimenters can use *any* software to provide vehicle control. Even the choice of programming language is open, and an experimenter familiar with a different vehicle programming framework than the ones we provide can use it to control the AERPAW vehicles. The only requirements we have on the vehicle control application is that it needs to produce MAVLink (version 2) messages that comply with the rules of the MAVLink filter detailed in Section 4.1.

However, to facilitate the introduction of a new experimenter to our testbed (especially if they have not previously used MAVLink) the AERPAW team is providing several application frameworks that can be used to develop vehicle control applications.

As shown in Fig. 8, at the bottom of the AERPAW recommended software stack is pymavlink [8]. Pymavlink is an open source python library that is primarily concerned with packing and unpacking MAVLink messages, and transmitting and receiving them through various connection types (primarily serial, TCP, and UDP). The library offers several examples and utilities demonstrating how to listen for a particular MAVLink type of message, and when received, how to process it. Our MAVLink filter is built on top of pymavlink.

Building on pymavlink, DroneKit [9] is offering an object oriented framework for interacting with the vehicle. In this framework, there is a background process that listens to an incoming stream of MAVLink status messages and populates the properties of an avatar of the vehicle, its properties being accessible through this avatar (e.g., vehicle.attitude returns the attitude of the vehicle). Similarly, method functions of the avatar can be used to send commands to the vehicle (e.g., vehicle.simple_takeoff(aTargetAltitude) will command the drone to takeoff at the specified altitude).

4.3 AERPAW Vehicle Library

The final framework that we support in AERPAW is *aerpawlib* [10], a custom Python 3 library that has been developed specifically for AERPAW experiments, allowing an experimenter to capture radio measurements while issuing vehicle commands. The framework is built on top of the Python asyncio framework, which allows different concurrent processes to run simultaneously either synchronously, or asynchronously.

As far as the experimenter is concerned, using the *aerpawlib* API involves writing a “Runner”, which is a collection of hooks stored within a Python class. Each hook is implemented as an experimenter-provided function wrapped in a decorator that registers relevant information about how to call the function and any extra high level information about how the function should be run. The Runner class then uses reflection to examine, extract, and then bind any registered hooks within it. Because of this, when a Runner script is executed, it must use the *aerpawlib* package’s provided tooling. The tooling is able to examine a Python source file, find and extract any Runners present, and then parse them to determine how to control the flow of an experiment.

The primary Runner of *aerpawlib* provides a way for experiments to be expressed as state machines. Most experiments on the AERPAW platform can be broken down into a basic state machine. As an example, an experimenter who wants to have a drone go between several locations while taking measurements can express their experiment as a collection of states with each state corresponding to a different location.

Many experiments need to run multiple pieces of logic at the same time (e.g., moving a drone while actively taking radio measurements); *aerpawlib* makes heavy use of Python’s asyncio library and re-expresses some of its constructs in a simple way to encourage experimenters not familiar with Python to use them. In addition to managing asynchronous tasks, *aerpawlib* provides custom logic and oversight that runs at the same level as the experimenter’s code (e.g., automatic management of autopilot state and landing of vehicles at the end of the experiment).

Finally, *aerpawlib* has native support for constructs used by most experiments, such as GPS coordinate processing and processing of waypoint files that can be generated by most ground control software such as QGroundControl.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented a broad introduction to the hardware and software choices for AERPAW vehicles and portable nodes. Our vehicles are fully programmable, allowing experimenters to precisely control the trajectories of the portable nodes, thus leading to reproducible experiments. In the near future, on the hardware front, we plan to support parachutes for our UAVs, develop a battery management system, as well as a thrust stand, and a battery testing stand. We also plan to develop a new rover capable of handling rough terrain. Next version of AERPAW’s vehicle control software will support multi-vehicle scripts that safely allow for *coordination* between multiple vehicles.

6. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation PAWR Program under Grant No.: CNS–1939334.

7. REFERENCES

[1] G. Bielsa, M. Mezzavilla, J. Widmer, and S. Rangan, “Performance assessment of off-the-shelf mm wave radios for drone communications,” in *2019 IEEE 20th International Symposium on” A World of Wireless, Mobile and Multimedia Networks”(WoWMoM)*, pp. 1–3, IEEE, 2019.
[2] A. S. Abdalla, A. Yingst, K. Powell, A. Gelonch-Bosch, and V. Marojevic, “Open source software radio platform for research on cellular networked UAVs: It works!,” *IEEE Communications Magazine*, vol. 60, no. 2, pp. 60–66, 2022.
[3] J. Buczek, L. Bertizzolo, S. Basagni, and T. Melodia, “What is a wireless UAV? a design blueprint for 6G flying wireless nodes,” in *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*, pp. 24–30, 2022.
[4] “The cube orange overview.” https://ardupilot.org/copter/docs/common-thecubeorange-overview.html.
[5] “The ArduPilot web page.” https://ardupilot.org.
[6] “The PX4 web page.” https://px4.io.
[7] “MAVLink - micro air vehicle communication protocol.” https://mavlink.io/en/.
[8] “Pymavlink github repository.” https://github.com/ArduPilot/pymavlink.
[9] “DroneKit SDK.” https://dronekit.io/.
[10] “AERPAW vehicle library github repository.” https://github.com/morzack/aerpawlib-vehicle-control.