ARDUPLANE BASED AVIONICS

KHALED HACHEM[1], ANDREI ION[2]

Abstract. This paper presents the development of a software application engineered to display the state of an aircraft on digital monitors in real time, providing critical flight information to the pilot. It was developed using the Qt framework for C++, taking advantage of the powerful producer-consumer architecture offered by Qt. The application acts as a TCP or UDP client, depending on the desired setup, receiving data from an ArduPilot device, which is connected to the same network, or directly wired to the host computer with an Ethernet cable, allowing for high speed communication between the devices. Data is parsed using MAVLink 2.0 headers generated with the official MAVLink header generation Python application, and the ardupilotmega XML file, ensuring complete compatibility between ArduPilot and the software module. To achieve fluid motion and animation in the QML components, the application uses a multithreading strategy, where a secondary thread running in the background is responsible for reading and parsing the data, while the main thread renders the graphical components. The received data is stored in shared memory, and is made thread safe by using the Qt multithreading functionality. By using Qt, the code can be compiled for both Windows and Linux environments, making the application extremely portable.

Key words: Avionics, MAVLink, ArduPilot, TCP, UDP, QT.

1. INTRODUCTION

In aviation, real time telemetry data is crucial for maintaining situational awareness and ensuring safety during flight. Modern flight controllers such as the CubePilot family of controllers use a multitude of sensors to measure raw data, and then process it to obtain incredibly accurate estimates of their current state. The main purpose of this information is automatically controlling the aircraft, but it can also be used for manual flight, by providing it to the pilot in real time. For this to be possible, the information provided must be captured, decoded, then displayed on digital indicators, providing visual cues to the pilot. While many EFIS applications already exist, most of them are do not support MAVLink, or are based on Android

[1] INCAS – National Institute for Aerospace Research "Elie Carafoli"
[2] National University of Science and Technology POLITEHNICA, Bucharest

Ro. J. Techn. Sci. – Appl. Mechanics, Vol. 71, N° 1, P. 93–104, Bucharest, 2026

DOI:10.59277/RJTS-AM.2026.1.06

which is not suitable for critical flight information. This paper presents the design and implementation of an embedded Linux based avionics display application tailored for integration with the CubePilot family of flight controllers. The remainder of this paper details the design, architecture, and implementation of this project.

2. ARDUPILOT AND MAVLINK

ArduPilot is an open source, highly advanced autopilot software suite used to control many vehicle types such as multicopters, helicopters, fixed wing aircraft, submarines, rovers, and more. The software is developed by a community of professionals and enthusiasts. The original repository for the software was created in 2009, and since then it has been developed by a team of engineers, academics, and computer scientists. Currently, it is installed in over a million vehicles world-wide, making it deeply tested and validated, which is why it is also used by large institutions and corporations, as well as colleges and universities, for development and testing.

Ardupilot uses the MAVLink (Micro Air Vehicle Link) communication protocol. It is a highly efficient, light weight, and open source protocol specifically designed for unmanned aircraft and vehicles. Created by Lorenz Meier, it has become the primary protocol used by flight control software for communicating with ground control stations, companion computers, and custom hardware.

MAVLink is a binary protocol, and its payloads pack numbers and states into raw bytes, instead of sending human readable text, making this protocol able to send a large amount of data over low bandwidth channels.

It is also very easy to use in software development. The reference libraries for MAVLink are written in C, and they are header only, making them light weight. They also do not require any dynamic memory allocation, making them safe.

MAVLink utilizes XML files to define its messages. Using the python written generator, these XML files can be converted into code libraries. There are different dialects of MAVLink available. ArduPilot uses a custom dialect that is built on top of the standard one, adding some extra specialized messages unique to its code.

3. HARDWARE

The module consists of 3 monitors, each used to display the essential flight information to the pilot. One of the monitors displays attitude, heading, velocity, and altitude. Another one displays motor speeds, fuel, and battery capacity, and the third monitor displays a map with a plane icon on it, representing the current location of the aircraft.

The software was designed to run in an embedded environment, but it also required enough processing power to be able to render and display the digital indicators. A simple microcontroller is unsuitable for this task. To solve this issue, three LattePanda Alpha computers were used, one for each monitor. These computers feature a powerful intel processor, along with an Arduino Leonardo chip, they also include and RJ45 Ethernet port, 3 USB ports, and a display cable port, making them ideal for this task.

The on-board Arduino microcontroller is perfect for connecting the computer to different buttons and potentiometers which may be used by the pilot to send commands to the CubePilot (Fig. 1).

An Ethernet switch is used to connect the 3 computers and the CubePilot to a private local network. Using cables only makes the connection secure, reliable, and fast, minimizing the risk of any data loss or cybersecurity attacks.

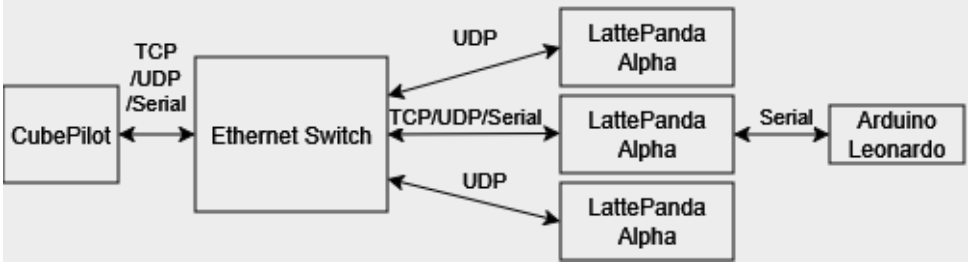


Fig. 1 – Network diagram.

4. QT AND C++

The computers for this hardware are capable of running a linux distribution as an operating system, which means that the application needed to be developed as a desktop app. At the same time, it needs to be able to process data in real time, and also display and render graphical elements.

One framework that handles all of these requirements is QT. QT is a C++ framework specifically designed for embedded applications and rendering graphical elements. Since it uses C++, it is very fast, making ideal for a real time application, and it also adds a vast amount of functionality to the programming language. QT provides a “producer-consumer” or event driven architecture. It also includes its own multithreading implementation, adds math functionality such as quaternions, networking capabilities for UDP and TCP servers, Serial data clients, and it provides QML, which is its own XML like scripting language for building graphical components.

The event driven architecture works beautifully in QT, especially when using multithreading. A class can have signals and slots, these are special functions. Slots work like a public function, they take data then they execute their logic. Signals are

emitted by an instance of an object when certain events happen. They provide data, and send it to a slot. This works even if the two objects live on different threads.

5. CLASSES AND DATA FLOW ARCHITECTURE

The application was designed to be able to capture data from the CubePilot using UDP, TCP, or Serial communication. One class was written for each of those methods, and by taking advantage of polymorphism, the main execution thread can create a client then decides which of the three classes to create an instance of and assign to that client depending on which argument is used when launching the app from the command line (Fig. 2).

The application also includes a UDP "repeater" class, which can also be enabled from the command line. It takes any and all data packets received by the client instance, then broadcasts them through UDP. Since only the main computer will directly receive data from the CubePilot, it will use the UDP repeater mode to provide that data to the other two computers.

Another important functionality of the app is the ability to read serial data from the Arduino Leonardo. The intel processor and the Arduino Leonardo are connected to each other through serial by default, this is due to how the LattePanda Alpha was designed. The Arduino runs a script that captures pilot inputs, which are transmitted to the app using an instance of the PilotInput class.

All of the data is handled by the MAVLink parser class. It is responsible for taking data from the client, whether that client is TCP, UDP, or Serial, the decoding that data. It also has the ability to encode commands using MAVLink and send it back to the client, which will then send it to the CubePilot. The PilotInput class receives data from the pilot, then emits it to the MavlinkParser, enabling the pilot to directly send commands to the CubePilot.

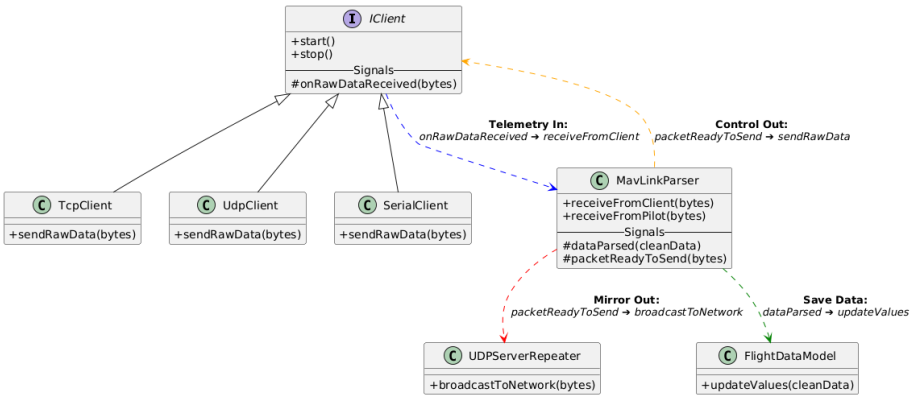


Fig. 2 – Class diagram.

6. FLIGHT MODES

While the application was mostly designed for displaying data, the CubePilot is capable of autonomous flight. Since the avionics application is directly connected to the CubePilot, and is capable of receiving pilot inputs, and sending them to the CubePilot, then it is only logical that it should be the module responsible for switching through flight modes.

The CubePilot is capable of many flight modes, but the most relevant for this project are the following:

- Takeoff
- Landing
- Guided
- Manual

A button above the central display is responsible for cycling through modes. A short press switches between Guided and Manual, while a long press switches between Takeoff and Landing. The landing and takeoff mode are for VTOL vehicles only.

To takeoff, the CubePilot needs to be armed, which requires the Extended Kalman Filter to be successfully initialized and running, correctly estimating the state of the aircraft. For Guided mode flight, GPS must be connected and receiving accurate data. Otherwise, only manual flight will be available.

When the pilot is ready to takeoff, they must perform a long press on the mode cycling button, this will set the internal mode of the application from idle to takeoff, send an arming command to the CubePilot, and finally send the takeoff command. Once in the air, the plane will be in the manual mode. If the pilot decides to switch to guided mode, the current altitude, course, and airspeed will be set as targets, and CubePilot will command the plane to stick to them. The pilot will be able to manually set the targets using a potentiometer and 3 buttons available above the central display. When switching back to manual, the set targets will be reset.

Once the pilot decides to land, he can either switch to Landing mode to execute an automatic VTOL landing, or perform a manual landing in the manual mode, the way a fixed wing plane typically does.

7. GRAPHICAL COMPONENTS

As previously mentioned, the module features three screens, each with a distinct window displaying different pieces of information. The three window were listed as the following:

- Main window – featuring:
    - Attitude indicator
    - HSI (Horizontal Situation Indicator)

  - Airspeed tape
  - Altitude tape
  - Climbrate tape
- Map window – featuring:
  - Map
  - Plane icon on the map
  - Ground speed
  - Heading
  - GPS connection state
  - GPS coordinates
- Fuel and motor information window – featuring:
  - Multirotor motor speeds
  - Main engine speed
  - Fuel level
  - Battery charge level

The application itself takes a command line argument specifying which window the app will be displaying. This argument will be set depending on which of the three computers the app is launched from.

The received information must be displayed in an easy to understand format. This is where QML shines, it enables building, rendering, and displaying graphical components. The graphical components were individually built then added to the window components.

The graphical components were built using relative sizes, making them able to fit on in any window, while the windows were designed to use an absolute size equal to the physical size of the monitors, ensuring that each window uses the full monitor, not wasting any free space, this is important since the monitors are not particularly large, and there is a lot of information which needs to be conveyed to the pilot.

The components displayed in each window are the following:

1. Main window:
   a. Attitude indicator

The attitude indicator QML Component is used to display Roll and Pitch. The blue space represents the sky, while the green space represents the ground. The white central line separating the two represents a pitch of 0 degrees, and the vertical ticks present different levels of pitch, starting at 0 degrees, with an increment of 2.5 degrees (Fig. 3).

The ticks on the upper end of the screen represent roll, the ticks show a roll of 0, 10, 20, 30, 45, and 60 degrees.

The little rectangle below the triangle is the slip indicator, showing how much the plane is drifting during rolling maneuvers. Slip is calculated as the lateral acceleration of the aircraft.

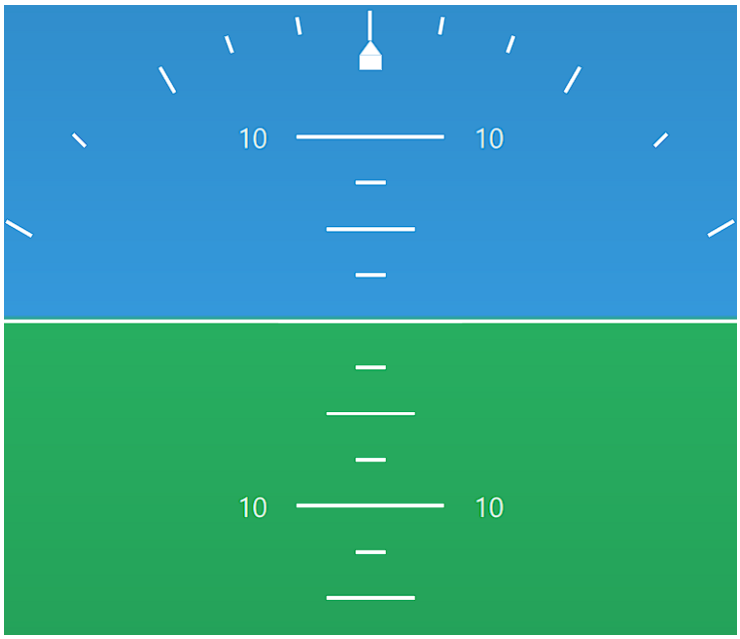


Fig. 3 – Attitude indicator component.

b. HSI (Horizontal Situation Indicator)



Fig. 4 – HSI QML component.

The two cards in the upper corners display the heading and course. Here, heading refers to the current yaw of the plane, as in where the nose of the plane is looking, while course tells the pilot where the plane is actually flying towards, this depends on GPS data (Fig. 4).

The purple needle inside of the compass represents the course, while the upside-down purple triangle indicates the yaw. The yellow needle represents the bearing pointer, which always points to the target coordinates stored inside of the CubePilot, these coordinates are usually the coordinates where the vehicle is supposed to land. The gray dots are used to display the deviation from the target course. The orange symbol in the compass is the heading bug, displaying the direction that the pilot wants to fly towards. This value of the heading bug is manually selected by the pilot, or automatically selected when the CubePilot goes from manual mode to guided mode.

c. The main window (Fig. 5)

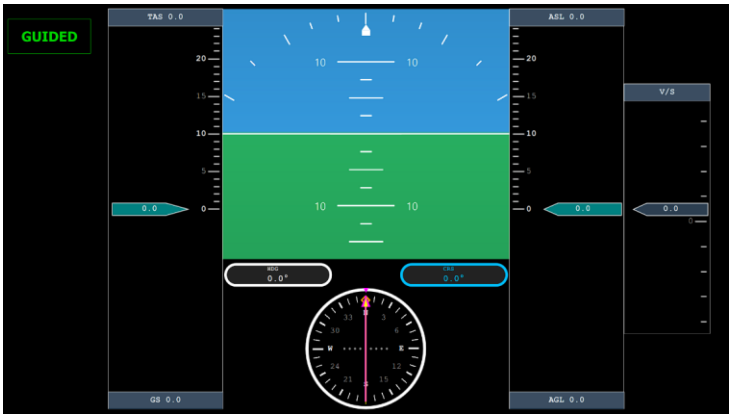


Fig. 5 –The complete main window.

The three previously mentioned tapes – Altitude, Airspeed, and Vertical Velocity – can be seen present next to the Attitude indicator and the HSI components.

The altitude tape, present on the left has two rectangles displaying GS and TAS. GS stands for GroundSpeed, it represents the speed of the aircraft relative to the ground, and it is provided by the GPS module. TAS stands for True Air Speed, it represents the speed of the aircraft relative to the air around it, and is important to the pilot, since a low airspeed can lead to the stalling of the vehicle. The value on the central indicator of the tape is the same as the TAS.

On the right side of the window, the altitude tape is present. ASL stands for Altitude Sea Level, and it represents the altitude relative to the sea level, while AGL stands for Altitude Ground Level, and it simply refers to the altitude relative to the takeoff point. This is self explanatory. The central indicator inside of the tape displays AGL.

The rightmost tape represents the climbrate of the vehicle, and V/S stands for vertical speed.

In the upper left corner, a green box can be seen. That box displays the current mode that the CubePilot is currently operating in. In the future, more boxes such as this one will be added, such as the arming state of the CubePilot, GPS fix information, and other critical information.

2. Fuel and motor information window (Fig. 6)

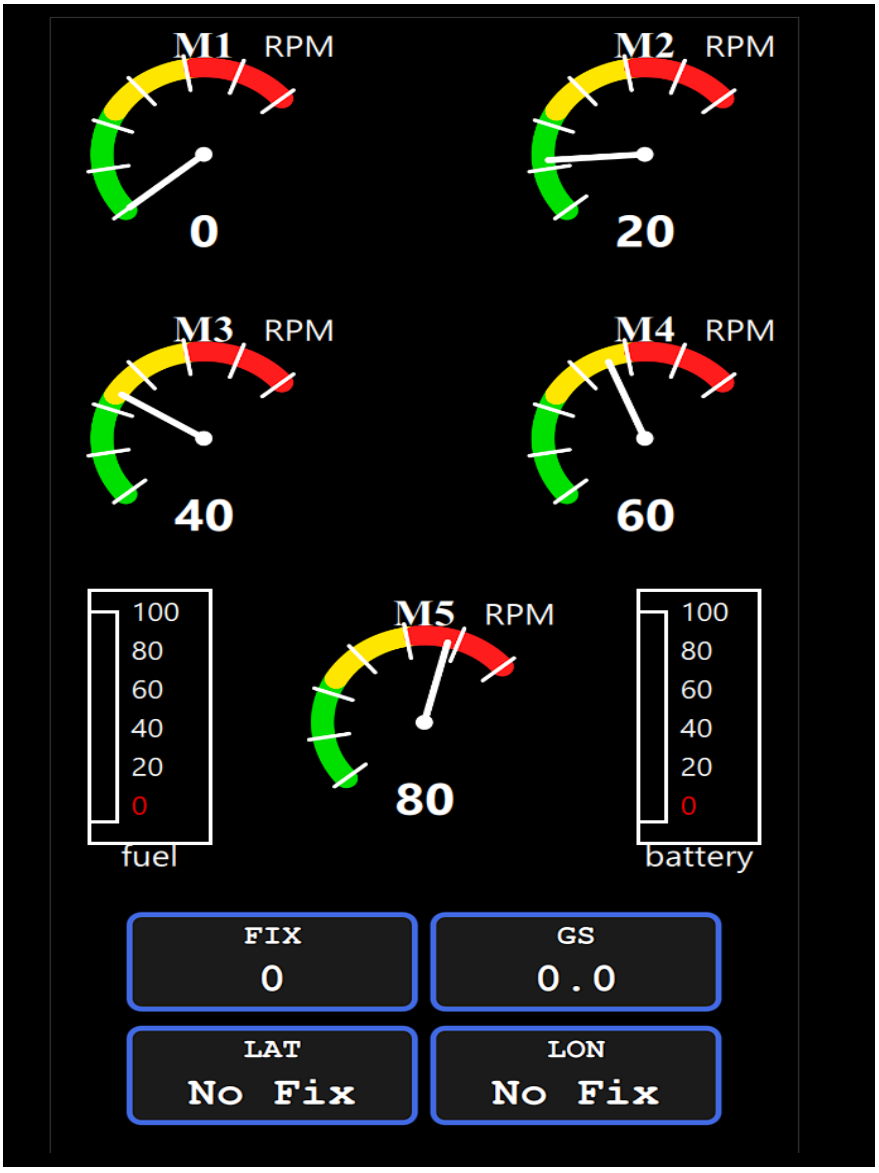


Fig. 6 – Fuel and motor information window.

This window displays the current fuel and battery charge levels, these are crucial for flight. The pilot must know at all times whether he has enough fuel to keep flying, or they should perform a landing sequence.

The 5 gauges are used to display the current RPM levels of the motors. The 4 upper gauges numbered M1 through M4 are for the multirotor VTOL electric motors, while M5 displays the combustion engine speed.

In the 4 boxes below, the GPS fix state can be seen, along with the ground speed, latitude, and longitude (Fig. 7).

3. Map window



Fig. 7 – Map window.

The map window displays the plane icon above a set of map tiles. These map tiles were downloaded using a specialized tool called SAS Planet. The current tiles downloaded are from GoogleMaps, and the amount of tiles is small. They were

only downloaded for testing purposes, and will soon be changed to a larger set of tiles provided by OpenStreetMaps.

The map window features a dynamic zoom level adjusting itself automatically depending on the ground speed of the aircraft. A common issue with this setup occurs when the vehicle is flying at a speed which sits very near to the border of the next zoom level, which may lead to the zoom level flickering and constantly changing. The solution to this is using a Schmitt trigger mechanism, where the required speed to move to the next zoom level is higher than the speed required to go back to the previous level.

The boxes in the lower end of the window Display the GPS Fix, Ground Speed, current map zoom level, Latitude, Longitude, and Heading.

The three windows were tested on the displays to ensure that all of the elements were properly sized and scaled. The map tiles were not present on the disk of the computer responsible for displaying the map at the time of the test, which is why they were not yet visible (Fig. 8).



Fig. 8 – Display test on the avionics module.

8. CONCLUSIONS

The avionics module allows the pilot of an aircraft to monitor and quickly interpret critical flight data such as attitude, altitude, heading, velocity, and more. This information is what allows the pilot to safely operate the vehicle, allowing it to reach its detination and fullfil its mission.

At the same time, the module acts as an interface between the pilot and the CubePilot device, enabling the use of the CubePilot's advanced autonomous

control systems. While the ArduPilot software stack was initially developed for controlling unmanned aerial vehicles, it has since proven itself to be one of the most reliable, safe, and rigorous open source flight computer pieces of software written. Developed by a large community of proffessionals and enthusiasts, and tested by a worldwide community of hobbyists, it offers exceptional state estimation, flight control capabilities, and software documentation.

While still in a prototyping stage, this avionics module should become part of a much more advanced, robust, mature, and complete device that will ultimately serve pilots in real aerial vehicles.

Acknowledgements. This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CCCDI – UEFISCDI, project number PN-IV-P7-7.1-PTE-2024-0250, within PNCDI IV

Received on June 15, 2026

REFERENCES

1. ArduPilot Development Team, ArduPilot Documentation, ArduPilot Project, 2025. Available at: https://rdupilot.org.
2. MEIER, L. *et al*., MAVLink Micro Air Vehicle Communication Protocol Specification, 2024. Available at: https://mavlink.io.
3. The Qt Company, Qt QML Reference Documentation, Qt Group, Oslo, Norway, 2025. Available at: https://doc.qt.io/qt-6/qmlapplications.html.
4. The Qt Company, QML Reference Manual, Qt Group, Oslo, Norway, 2025. Available at: https://doc.qt.io/qt-6/qmlreference.html.
5. ÇİĞDEM GÜNEŞ, *Human Factors Issues of Glass Cockpit Automation* (Chapter 2. "Glass Cockpit Aircrafts"), The Graduate School of Natural and Applied Sciences of Middle East Technical University, Ankara, Turkey, 2010. Available at: https://etd.lib.metu.edu.tr/upload/3/12611791/index.pdf.