SWARM Flight Stack & Simulation Research

-Kshitij Nigam

This paper presents a technical study of open-source drone flight control stacks, with a comparative analysis of ArduPilot and PX4 and an in-depth examination of Software-In-The-Loop (SITL) simulation for safe UAV development and testing.

Introduction

In the last decade, there has been a significant increase in the use of unmanned aerial vehicles (UAVs). This surge is attributed to technological advancements, such as the miniaturization of components, the availability of GPS systems and high-resolution cameras, among others, which have made drones more affordable for civilian use

An autopilot is an embedded computing system that executes software responsible for the control and guidance of unmanned aerial vehicles (UAVs). Its core functions include stabilizing the platform to maintain attitude and heading, executing mission plans through predefined waypoints, initiating failsafe procedures when required, and communicating with ground control stations for telemetry monitoring and data management.

Over time, numerous autopilot systems have been developed for civilian, military, research, and educational purposes. Certain autopilots are designed as plug-and-play solutions, targeting users who do not need detailed insight into internal programming or system architecture. In contrast, open-source autopilots allow developers to modify internal functionality, with both hardware and software designs made publicly available. The importance of open-source autopilots stems from their flexibility, cost-effectiveness, and strong community-driven support, which together promote innovation, experimentation, and customization.

Autopilot	Year	Country	Discontinuity
Paparazzi	2003	France	Active
MatrixPilot	2008	USA	2016
APM (ArdupilotMega)	2009	USA	Active
MultiWii	2010	USA	2016
PX4 (Pixhawk)	2011	Switzerland	Active
Baseflight	2012	USA	2014
TauLabs	2012	USA	2016
OpenPilot	2013	USA	2015
Cleanflight	2014	USA	2020
LibrePilot	2015	USA	Active
Betafilght	2015	USA	Active
dRonin	2015	USA	2019
iNAV	2016	USA	Active

The table summarizes the timeline, origin, and current activity status of major open-source drone autopilot systems.

Autopilots

-an autopilot can also sometimes be referred to as the flight controller in autonomous drones

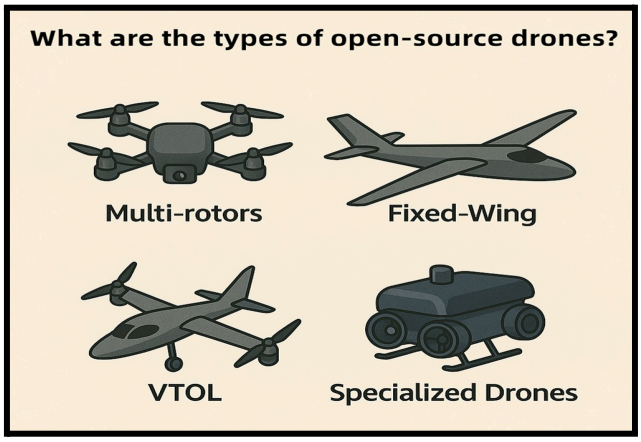
Flight controller - hardware
Autopilot - software (needed in the hardware)

UAV autopilot systems enable unmanned aerial vehicles to execute complete missions autonomously without continuous manual remote control. Such missions may include cargo delivery, aerial mapping, surveillance, and other operational applications. Mission parameters are defined by operators through a ground control station, after which the autopilot autonomously guides and controls the UAV to accomplish the assigned task.

How Does an UAV Autopilot Work?

By integrating data from multiple sensors such as gyroscopes, accelerometers, magnetometers, and GPS, the flight control processor estimates the UAV's attitude, velocity, position, and heading. Using this information, the autopilot controls the flight and operation of the vehicle in accordance with user-defined parameters. Additional sensors can further enhance the system by enabling obstacle detection and collision avoidance during operation.

Fully autonomous UAVs are capable of executing complete flight plans, including vertical take-off and landing (VTOL) or runway-based take-offs, in-flight manoeuvres, and landing procedures. In addition to fully autonomous operation, many UAV autopilot systems also provide manual and assisted-manual flight modes, allowing operators to retain partial or full control when required. To enhance reliability and safety, most UAV autopilots incorporate single or multiple redundancy mechanisms that enable continued operation in the event of system failures.



UAV autopilot systems are designed to support a wide range of vehicle types, including fixed-wing aircraft, multirotors, helicopter drones, parafoils, blimps, and tethered drones. Some manufacturers employ a unified hardware and software platform capable of operating across these vehicle classes through configurable flight phases and control channels, while others utilize common hardware paired with vehicle-specific software implementations.

An autopilot operates as a closed loop control system that continuously senses the UAV's state, on the basis of this data the autopilot makes decisions on the basis of what is required, these requirements are determined on the basis of set parameters by the user.

The steps:

1. **Sensor data acquisition -** The autopilot collects data from onboard sensors such as the IMU, GPS, magnetometer, and barometer to measure the UAV's motion and position.
2. **State estimation -** Sensor data is fused, typically using an Extended Kalman Filter, to estimate the UAV's attitude, velocity, position, and heading.
3. **Guidance and Navigation-**Using mission parameters and flight modes, the autopilot determines desired motion targets such as waypoints or velocities.
4. **Control System -** Cascaded control loops convert the desired motion into actuator commands while maintaining stability.
5. **Actuator Output -** Control signals are sent to motors and servos to execute the commanded flight.
6. **Safety and Failsafes -** The system continuously monitors health parameters and triggers predefined safety actions if failures occur.
7. **Communication -** The autopilot exchanges commands and telemetry with ground control stations and companion computers.

ArduPilot

-A robust, open-source autopilot software platform that enables autonomous operation of unmanned systems—including multirotor and fixed-wing drones, surface vehicles, ground rovers, and underwater vehicles.

ArduPilot runs on the flight-controller and computed commands for the actuators based on sensor data, pilot inputs and mission logic.

1. Sensors (IMU, GPS, barometer, compass) tell ArduPilot the vehicle's state
2. ArduPilot decides how the vehicle should move (attitude, speed, position)
3. It sends low-level control signals (PWM/DSHOT,) to motors, ESCs, or servos
4. Rotors respond to those signals and generate thrust

PWM - An analog-style control signal where motor or servo command is encoded in the width of a periodic pulse sent from the flight controller to the actuator.

DSHOT - A digital motor control protocol where throttle commands are sent as discrete digital packets to ESCs, offering higher precision, faster response, and improved reliability.

DSHOT is usually preferred over PWM

Architecture

The architecture of ArduPilot follows a layered, modular design that separates hardware interaction, state estimation, control, and mission logic to ensure reliability, portability, and real-time performance.

1. Hardware Abstraction Layer (HAL)

A hardware abstraction layer is a software interface that hides hardware-specific details and provides a consistent API, enabling the same application code to operate across different hardware platforms.

- Provides a uniform interface to sensors, processors, buses, and IO
- Allows ArduPilot to run on different flight controllers without changing core logic

2. Scheduler & Real-Time Task Management

The scheduler and real-time task management ensure deterministic, priority-based execution of control and sensor tasks at fixed rates, enabling stable and safe UAV operation.

- Executes tasks at fixed frequencies (e.g., IMU at high rate, GPS at lower rate)
- Ensures deterministic timing for stability-critical control loops

3. Sensor Drivers

- Read raw data from IMU, GPS, barometer, compass, airspeed sensors, etc.
- Pass processed data to the estimation layer

4. State Estimation (EKF2 / EKF3)

- Fuses multi-sensor data using Extended Kalman Filters
- Estimates attitude, velocity, position, and sensor biases in real time

5. Control Stack

- Cascaded controllers:
    - Rate control
    - Attitude control
    - Velocity control
    - Position control
- Converts desired motion into actuator-level commands

6. Vehicle-Specific Logic

- Abstraction layer for multirotors, fixed-wing, VTOL, rovers, boats, and submarines
- Reuses common control and navigation code across platforms

7. Mission, Mode & Navigation Layer

- Handles flight modes.
- Executes waypoint missions and autonomous behaviors

8. Communication & Logging

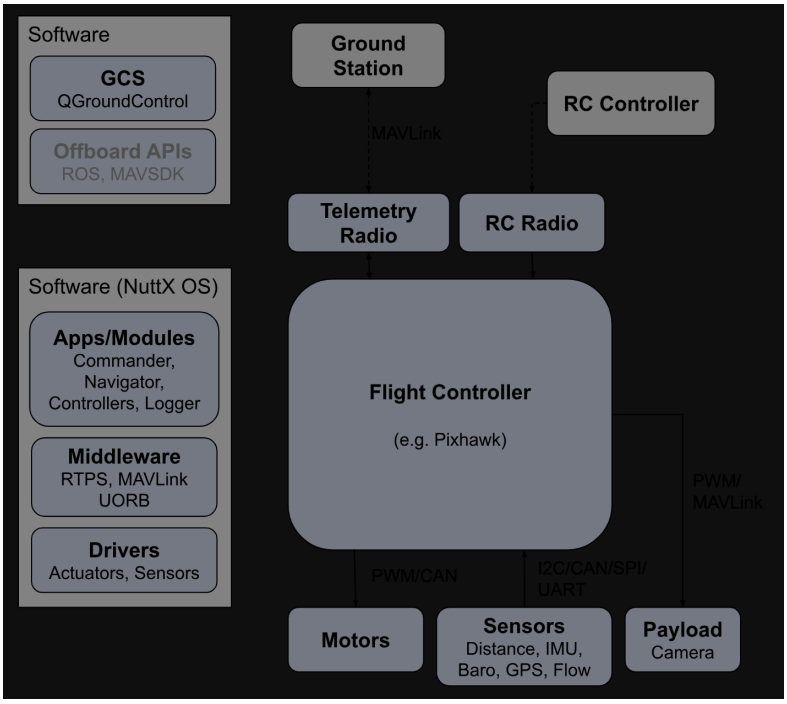
- MAVLink interface for telemetry and command exchange with ground stations
- Onboard logging for debugging, tuning, and post-flight analysis

—----------

PX4

-PX4 is designed as a modular system where many independent components (drivers, estimators, controllers, mission logic, communications) run concurrently and exchange data. The architecture cleanly separates middleware (core software infrastructure) from the flight stack (navigation and control logic). This makes the system scalable and reusable across different vehicle types.

Typical components:
These consist of a flight controller, GPS, external compass, manual controller and/or telemetry radio system, motors and/or other control actuators, payloads, and a power system.



Hardware Components

- **Flight controller** (running the PX4 flight stack), often including internal IMUs, compass, and barometer.
- **Motor ESCs** connected via PWM outputs, DroneCAN (DroneCAN allows two-way communication, unlike single-direction PWM), or other communication buses.
- **Sensors** such as GPS, compass, distance sensors, barometers, optical flow sensors, ADS-B transponders, etc., connected via I2C, SPI, CAN, UART, and other interfaces.
- **Camera or other payload**, which may be connected through PWM outputs or via MAVLink.
- **Telemetry radios** for communication with a ground station computer or software.
- **RC control system** for manual pilot input.

uORB Messaging System

ORB is an asynchronous publish–subscribe messaging API used for inter-thread and inter-process communication within the PX4 system.

uORB is implemented in the uorb module and is started automatically early in the PX4 boot sequence using uorb start, as many system applications depend on it. Unit tests for uORB can be executed using uorb_tests.

EKF2 State Estimation

PX4 uses the Extended Kalman Filter 2 (EKF2) as the default state estimator to fuse measurements from IMU, GPS, barometer, magnetometer, and other sensors into accurate attitude and position estimates. EKF2 provides statistically rigorous estimates used by the flight controllers.

Flight-Mode State Machine

PX4's *command/state machine* resides in the commander logic and governs flight modes (Manual, Stabilized, Altitude/Position control, Auto, RTL, Land, etc.). It keeps the vehicle in a safe operating state and manages transitions based on pilot input, mission progress, or failsafe conditions.

QGroundControl

QQGroundControl is the primary ground control station for PX4, communicating via MAVLink to configure the vehicle, manage flight modes, plan missions, monitor telemetry, and download flight logs. The usual process :

1. **MAVLink connection**: PX4 exchanges telemetry and commands with QGroundControl using MAVLink.
2. **Setup & configuration**: Sensors, airframe, RC, and safety parameters are calibrated.
3. **Mode & mission control**: Flight modes are selected and waypoint missions are uploaded.
4. **Monitoring & logs**: Live status is displayed and flight logs are downloaded for analysis.

PX4 vs ArduPilot

- **PX4** treats the autopilot as a **distributed system** where independent modules exchange data through messages.
- **ArduPilot** treats the autopilot as a **centralized real-time control system** with strong safety guarantees.
- PX4 prioritizes **external autonomy (ROS 2)**, while ArduPilot prioritizes **onboard autonomy and redundancy**.
- PX4's architecture is closer to modern robotics software stacks; ArduPilot's is closer to embedded safety-critical systems.

PX4 emphasizes modular, message-based architecture and tight ROS 2 autonomy integration, whereas ArduPilot emphasizes a robust, layered real-time architecture focused on safety, reliability, and broad vehicle support.

Aspect	PX4	ArduPilot
Core Design Philosophy	Strongly modular, message-driven architecture optimized for research, autonomy, and modern middleware	Monolithic but modularized architecture optimized for reliability, safety, and wide vehicle support
Software Architecture	Independent modules communicating via internal message bus	Layered architecture with tightly integrated subsystems
Internal Communication	**uORB** publish–subscribe messaging system	Direct function calls + internal data structures
State Estimation	**EKF2** (default), tightly integrated with PX4 middleware	**EKF2 / EKF3**, selectable with extensive redundancy support
Flight-Mode Handling	Explicit **flight-mode state machine** managed by commander logic	Mode-based control logic with strong failsafe coupling
Real-Time Scheduling	Module-level scheduling with message-triggered execution	Central scheduler with fixed-rate task execution
Hardware Abstraction	Clean separation via drivers and middleware	Strong **HAL** enabling broad hardware compatibility
ROS 2 Integration	Native, first-class **ROS 2 autonomy integration** (uXRCE-DDS, offboard & external modes)	ROS supported mainly via MAVLink and companion computers
Primary Strength	Autonomy research, ROS 2 integration, clean modularity	Stability, maturity, mission reliability, multi-vehicle support
Learning Curve	Steeper (architecture-heavy, middleware concepts)	Easier for beginners, extensive documentation
Typical Use Cases	Academic research, autonomy stacks, companion-computer workflows	Commercial drones, long missions, field-tested deployments

Software in the loop (SITL)

Software-In-The-Loop (SITL) is a simulation approach in which the actual autopilot flight software runs on a desktop computer while the vehicle, environment, and sensors are simulated by a physics engine. The autopilot executes exactly the same estimation, control, and mission code as it would on real hardware.

The SITL basically tests the software's ability to make decisions in multiple parameters.

Why SITL is critical?

- Eliminates risk of damaging airframes, sensors, or motors
- Enables early validation of flight modes, control logic, and autonomy
- Allows repeatable testing under identical conditions
- Accelerates development before hardware integration

What is and is not connected in SITL

- No physical drone is connected
- No real motors, sensors, or airframe are used
- Only the autopilot software is running
- The software is fed simulated sensor data
- Its outputs are sent back to the simulator, not to real hardware

AS APPOSED TO HITL - Hardware in the loop, usually the flight controller is connected to the sim, this makes the outcomes more accurate.

SITL is a core development and testing tool in both **PX4** and **ArduPilot** ecosystems.

Workflow: Data flow in a SITL simulation

1. **Physics engine (e.g., Gazebo)** simulates vehicle dynamics and environment
2. **Simulated sensors** (IMU, GPS, barometer, magnetometer) are generated from physics
3. **Autopilot SITL instance** receives sensor data as if from real hardware
4. **State estimation and control loops** (EKF, attitude/position controllers) run unchanged
5. **Actuator commands** (motor outputs) are sent back to the simulator
6. **Simulator updates vehicle motion**, closing the loop in real time