Technical Overview

Change log

Date	Changes
2024-11-05	Created file
2024-11-11	Initial version
2024-11-26	Added architecture diagrams
2024-11-27	Added architectural perspectives and updated mindmap

Introduction

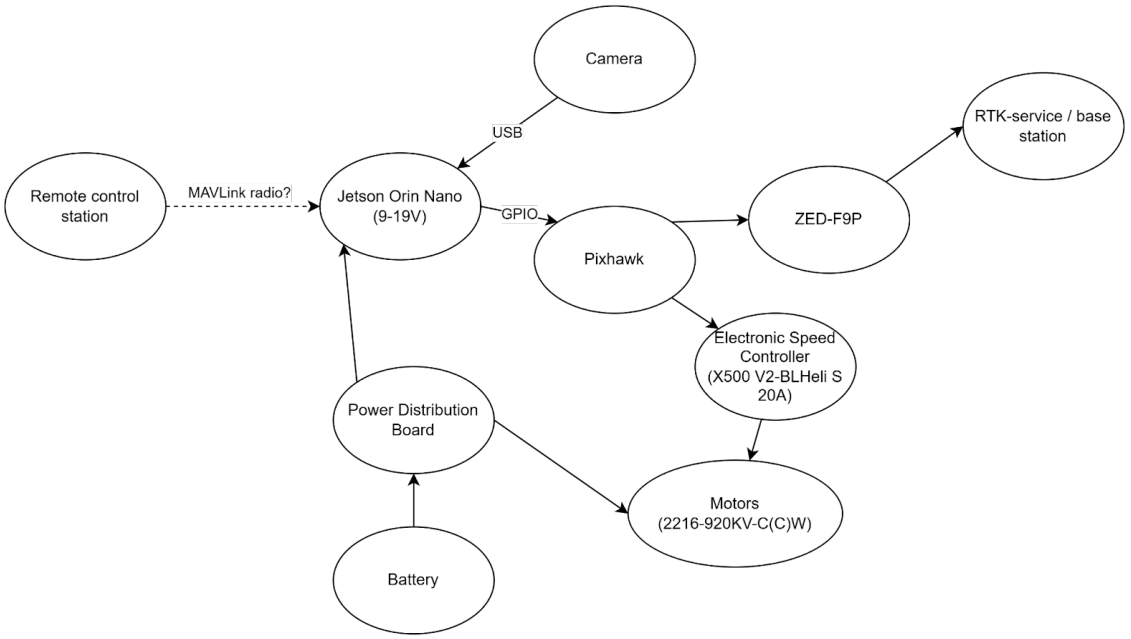
This document provides a comprehensive overview of our project, detailing the system's architecture, key technical decisions, progress so far, and future plans. Our goal is to develop an autonomous drone system by integrating permissive open-source components using Robot Operating System 2 (ROS2). The system is designed to run on an NVIDIA Jetson Orin Nano (or NX), interfacing with a Pixhawk flight controller via PX4 APIs and MAVLink protocols. We are leveraging simulation environments such as Gazebo for testing before conducting physical flights.

Project Overview

The primary objective of this project is to develop an autonomous drone capable of object detection under safe and efficient operating conditions. The system will integrate several hardware and software components to achieve autonomy, real-time data processing, and robust communication between subsystems. This includes integration of sensors and hardware components with the help of ROS2. Applying safety mechanisms for manual override and testing and validating the system in simulated environments before physical deployment.

System Architecture

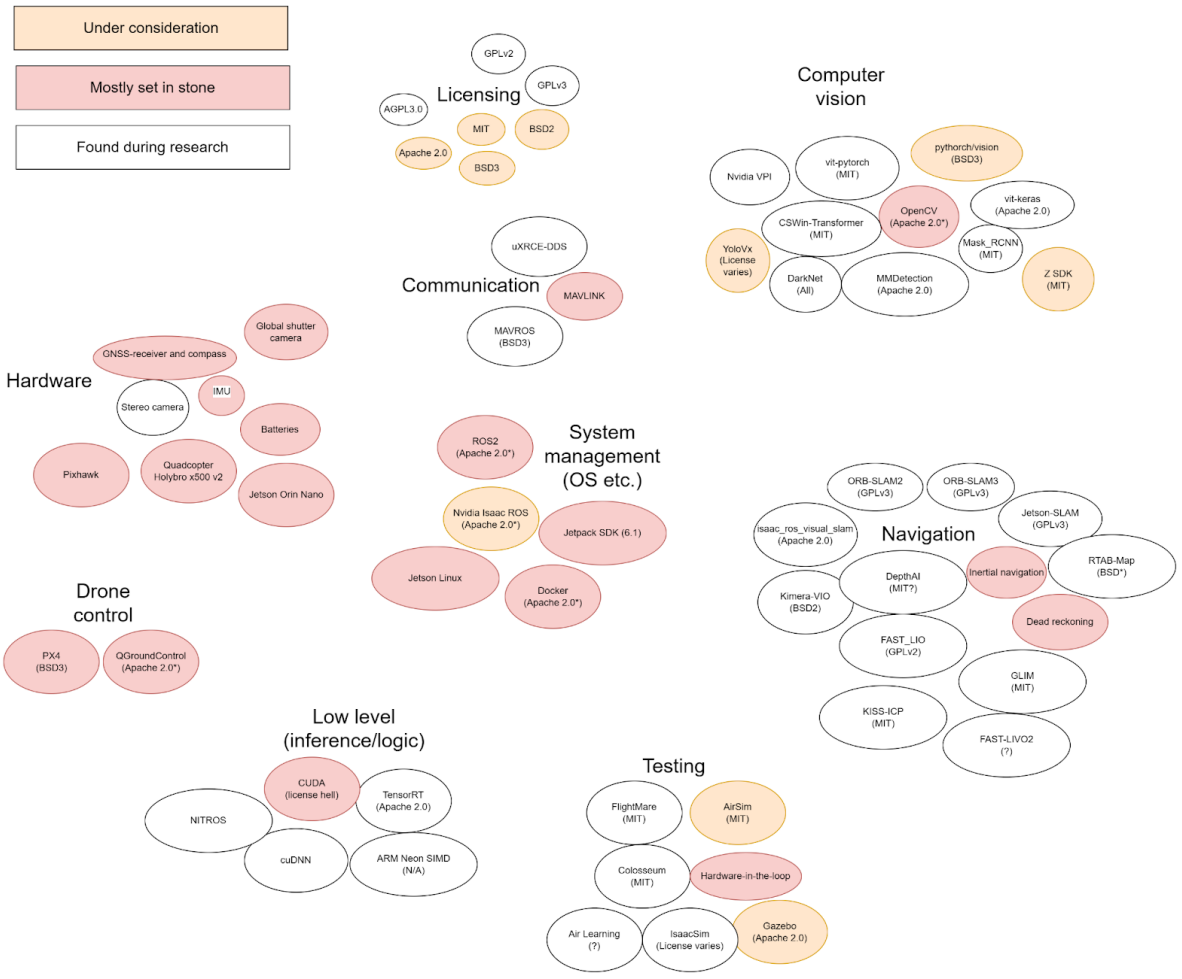
Below, you can find the high-level overview of our hardware system. It includes the building components and the communication links.



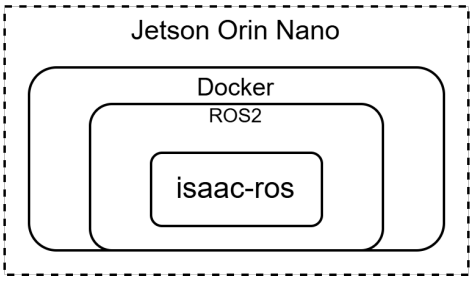
The Pixhawk serves as the autopilot, receiving inputs from the GNSS receiver (ZED-F9P) and the electronic speed controller. The system is designed to track the drone using a remote control station, and in case of errors, there should always be the option to switch to manual mode for controlling the drone. This ensures the drone can land safely, even if software bugs are encountered.

The Jetson Orin Nano is responsible for performing the most computationally intensive tasks. As shown in the diagram, the power supply is connected to the Jetson, which then distributes power to other system modules.

We agreed with the PO to create a mind map (shown below) that highlights all the libraries and technologies considered so far, as well as those selected for the project. Three colors are used to indicate our decisions:



In addition, we started working on the software architecture of our system but it is not ready yet:



Development Progress

During the first two sprints (0 and 1) we tried to set up a working environment. It included making software-in-the-loop (SITL) and hardware-in-the-loop (HITL) simulations, setups of ROS2 and Jetsons, and flashing custom programs to Pixhawk 4 devices. Most of us did not have any experience with those, so all of us learned many new things, mainly how to set different environments and fix errors that arise during this process. Originally we considered using MAVLink for the communication between Pixhawk device and Jetson Orin Nano, but

uXRCE-DDS proposed by PX4 community so far proved to be a better choice, as the instructions for its usage are well-written and easy to follow.

Significant Decisions

At the beginning we planned to use the AirSim simulator, but at this moment Gazebo seems to be a more suitable simulator for our project since Gazebo is still supported compared to AirSim and there is good documentation on how to set it up. In the future, we might still have some issues with the Gazebo Harmonic as we probably will need HITL for testing, but the official PX4 documentation mainly contains information about Gazebo Classic for this purpose, which is deprecated in the current PX4 versions. If HITL won't work on Gazebo Harmonic, we can switch to jMAVSim, which is also a PX4 compatible simulator but it can come with the cost of available physics in the simulation.
Another significant decision is that we have chosen ROS2 instead of ROS1 due to a few aspects. First of all, ROS2 offers better performance, especially for multi-threaded and multi-process applications, which is crucial for real-time operations on the drone. Secondly, it introduces new features such as support for Data Distribution Service (DDS) that we will be using, when working with Pixhawk and Jetson. Thirdly, ROS2 is actively maintained and seems to be the future direction of the ROS ecosystem. Furthermore, Nvidia has a ROS2 based robotics ecosystem called Isaac ROS which may want to be utilized as it provides hardware acceleration features for Jetson Orin devices. These ensure long-term support and community engagement.

Architectural Perspectives

This website presents 7 different viewpoints: Context, Functional, Information, Concurrency, Development, Deployment, Operational. To comprehensively understand the system, we examine it through the following two viewpoints:

Information View

Sensor data flows from the sensors to the Jetson for processing and creating a command for the Pixhawk on what it should do next. After this these processed commands are sent from the Jetson to the Pixhawk and the autopilot modifies the parameters of the motors to apply those commands. Then the telemetry data is sent from the Pixhawk to the GCS. Logs and sensor data can be stored on the Jetson for post-flight analysis.

Concurrency View

Sensor data acquisition and processing run concurrently with navigation computations to optimize performance and responsiveness. Simultaneously, communication with both the Pixhawk and the Ground Control Station (GCS) occurs in parallel to maintain real-time operational capabilities. ROS2's multi-threaded architecture facilitates this concurrent execution, effectively managing multiple processes. Additionally, considerations for a real-time operating system (RTOS) are incorporated to handle time-critical tasks, ensuring that high-priority operations receive the necessary computational resources.