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
2025-02-24	Updated the following sections: Development progress, Significant decisions and Architectural perspectives
2025-05-05	Mentioned about containerization and write about the software architecture

Introduction

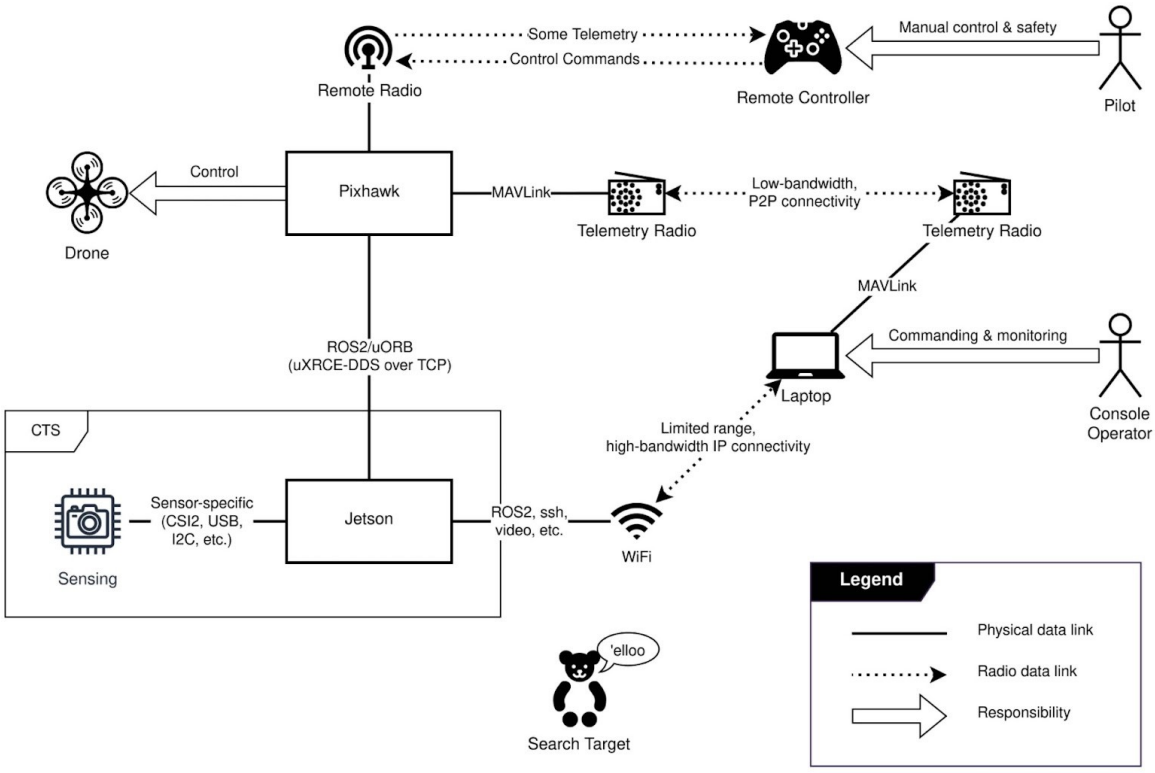
This document provides a comprehensive overview of our project, detailing the system's architecture, key technical decisions, progress, and possible future prospects. Our goal is to develop an autonomous drone system by integrating permissive open-source components using Robot Operating System 2 (ROS2). 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. In the actual drone the used computing module is Jetson Holybro Pixhawk Baseboard. 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.

Hardware Architecture

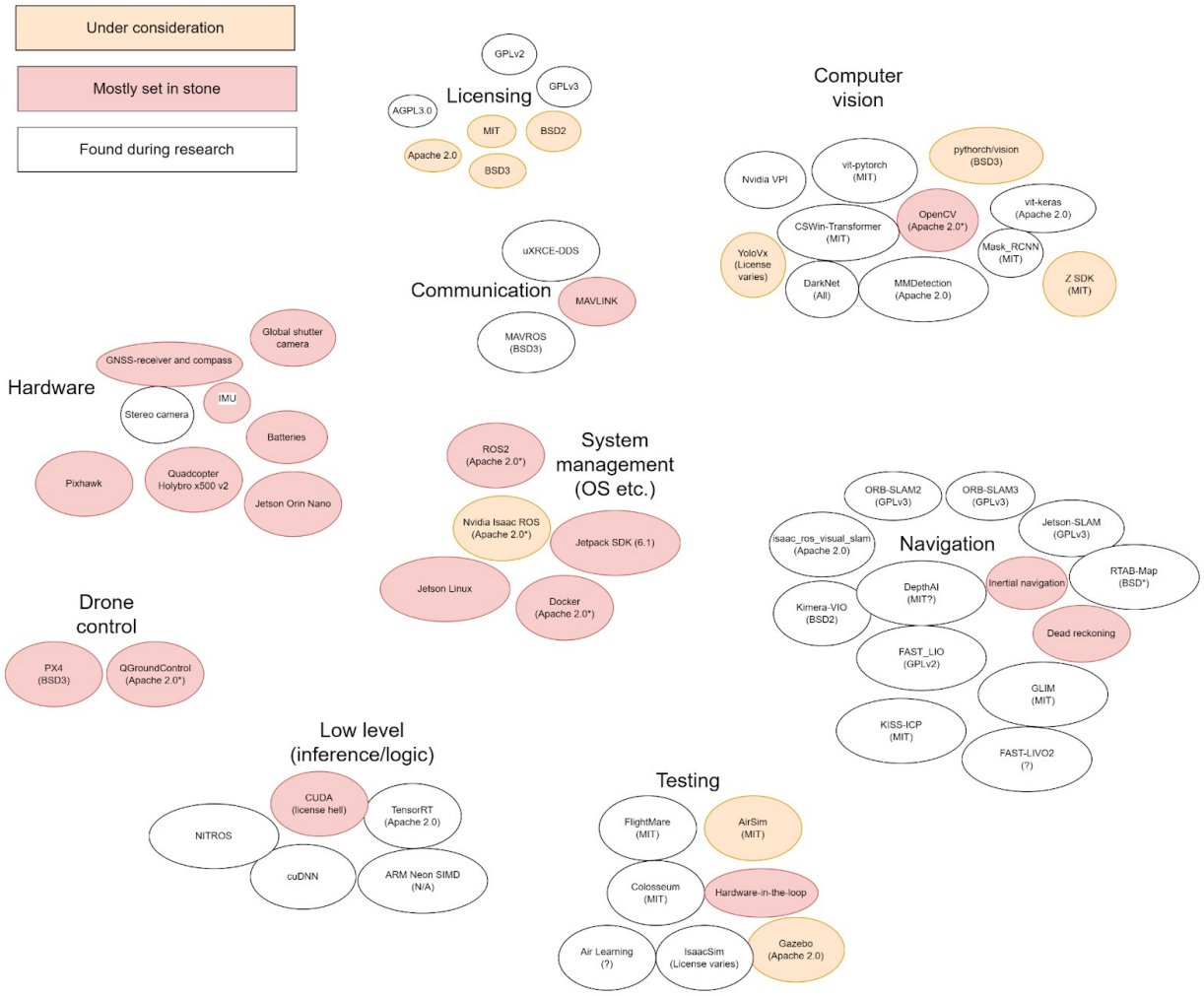
Below, you can find the general overview of our hardware system. It includes the building components and the communication links.



The Pixhawk module 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 (QGroundControl), 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.

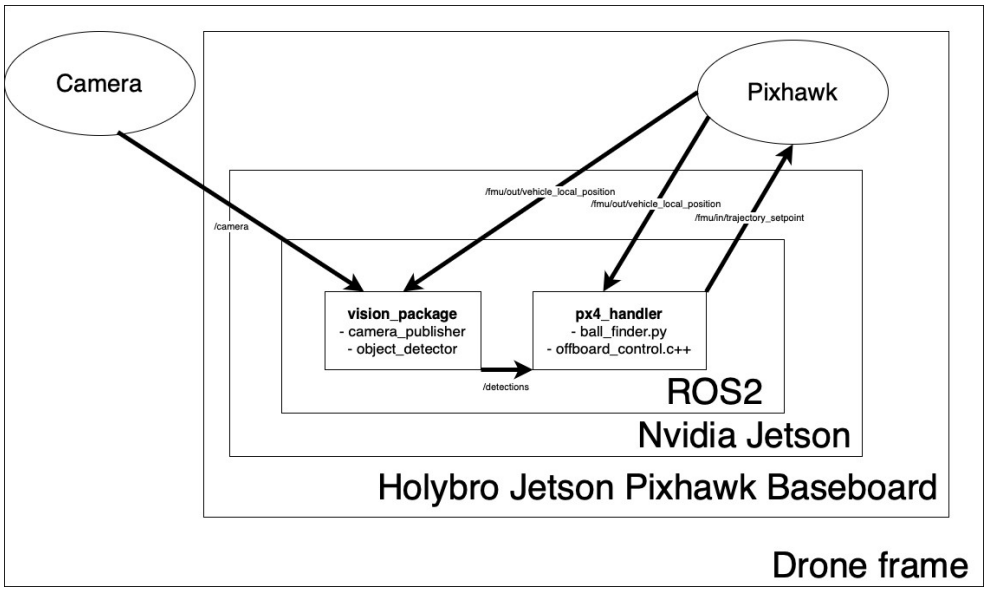
The Nvidia Jetson computer in the Holybro Pixhawk Jetson Baseboard is responsible for taking camera inputs and using them to locate the target, and then with the help of its ROS2 nodes to center the drone on top of the target.

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. These technologies are also applied in the software architecture and the code of our software.



Software Architecture

Below is a simple picture of our software architecture, including the ROS2 packages and their nodes, Pixhawk module, and a camera module.



Software System

The software system consists of a ROS2-workspace with two packages, each with two essential scripts:

1. **vision_package**
   (a) **image_publisher** Takes camera footage from a downward-facing camera and publishes it to /image_topic. Uses OpenCV.
   (b) **object_detector** Detects an object any sends its image pixel coordinates [x, y] to /detections. Utilizes Yolov5 CV model and uses PyTorch-library.
2. **px4_handler**
   (a) **ball_finder.py** First, starts surveying a determined area, for example an area inside 8m north and 12m east. Subscribes to /detections -topic, and if an object with wanted ID (corresponding to the ball) is detected, stops its current movement and centers itself on top of the ball, and descents. Two main functions inside this script:
      i. **move_to_waypoint** Takes a tracepoint [x, y, z] coordinates and moves to that coordinate. Is achieved by measuring the vector-distance to the target and comparing it to a given tolerance, and if not true then going towards target unit vector. Normally does surveying but if a ball is detected - exception.
      ii. **go_on_top** Takes an object center on camera and camera Height/Width. Formally:

      Image and ball centers rotated and normalized:

$$\vec{C} = \begin{bmatrix} \frac{W}{2} \\ \frac{H}{2} \\ 0 \end{bmatrix}, \quad \vec{B} = \begin{bmatrix} x_b \\ y_b \\ 0 \end{bmatrix}, \quad \vec{d} = \vec{B} - \vec{C}, \quad \hat{d} = \frac{1}{\|\vec{d}\|} \begin{bmatrix} -d_y \\ d_x \\ 0 \end{bmatrix}$$

      Yaw rotation by θ, and final position:

$$\vec{d}_{\text{rot}} = \begin{bmatrix} \hat{d}_x \cos\theta - \hat{d}_y \sin\theta \\ \hat{d}_x \sin\theta + \hat{d}_y \cos\theta \\ 0 \end{bmatrix}, \quad \vec{T} = \vec{P}_{\text{current}} + \vec{d}_{\text{rot}}$$

   (b) **offboard_control.cpp** Receives tracepoints from ball_finder.py and starts sending them on regular intervals to the PX4 topic /fmu/in/trajectory_setpoint. Is the first node started and initially just keeps the drone in the position where started.

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 the PX4 community proved to be a better choice, as the instructions for its usage are well-written and easy to follow.

During Sprint 2 the development slowed down due to the holidays. However, in Sprint 3, we made significant progress. One of the key milestones was successfully assembling different drone components and achieving our first indoor flight - since outdoor testing required official permission. Another major achievement was enabling the camera to send images to other ROS2 nodes during flight. Additionally, the offboard control example presented by the PX4

team was successfully implemented, and with some modifications, we managed to make the drone navigate from one corner of a square to another.

The biggest highlights of Sprint 4 were the first outdoor flight, where we collected and analyzed important telemetry data, and the ability to test vision packages using YouTube videos.

In Sprint 5, documentation pages were created and we managed to conduct autonomous flights. First we tested simple movements both in the simulator and in real life, such as going back and forth and rotating in place, but later we started working on the ball tracking functionality. This functionality includes three parts: a) ROS2 node taking the camera footage and transforming it to a suitable format (working), b) ROS2 node that processes the previous camera footage and detects objects, specifically sports balls, after which it calculates the object center pixel coordinates on the footage and publishes these into a ROS2 topic, c) ball_finder.py node that starts a survey of an area of 8m x 12m, and if the previous node detects a sports ball, this node stops and places itself on top of the drone with the calculations in the Systems Architecture -chapter. It is now working in the Gazebo simulator, but it is not tested yet in real environments, we will conduct later today a flight with the same functionality. Additionally, the main repository was restructured and cleaned up and a single Dockerfile was created to make it easier to set up the working environment. Moreover, we collected and analysed a lot of telemetry data that PO needed.

Moreover, we managed to create a working version of Docker development environment. Now we have a Dockerfile that a person can just run on their home computer, and this image takes care of pulling all the relevant repositories, modules, and packages. Moreover, it starts the simulation and the relevant processes so that one can easily start to write new code.

Significant Decisions

At the beginning we planned to use the AirSim simulator, but we decided to move forward with Gazebo since it is still supported compared to AirSim and there is good documentation on how to set it up. Recently, we decided not to proceed with setting up the HITL simulation since it would require significant time, and we currently have limited time remaining. Moreover, we haven't needed it so far. Instead, we focused solely on setting up the software-in-the-loop (SITL) simulation environment and ensuring it functions properly.

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 is straightforward to launch as a Docker container. It may be utilized as it provides hardware acceleration features for Jetson Orin devices and is very easy to set up. These ensure long-term support and community engagement.

Nvidia has developed several packages that could be useful for our project, such as Isaac VSLAM and the Optical Flow SDK. Since implementing all these functionalities from scratch would be tedious and time-consuming, especially given the limited time remaining, we plan to integrate these libraries into our final product. However, because of time limitations, we narrowed our project scope and decided not to move forward with obstacle avoidance functionality.

Another significant decision that we made is that we decided to use a blue ball instead of a kitten as our drone's target, because of moral principles. Moreover, since not all of us had a device with GPUs needed for the simulation, we decided that we could give access to our computers via Google Remote Desktop. Therefore, we used the team member's home computer to run the simulation and test our code there.

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 viewpoints:

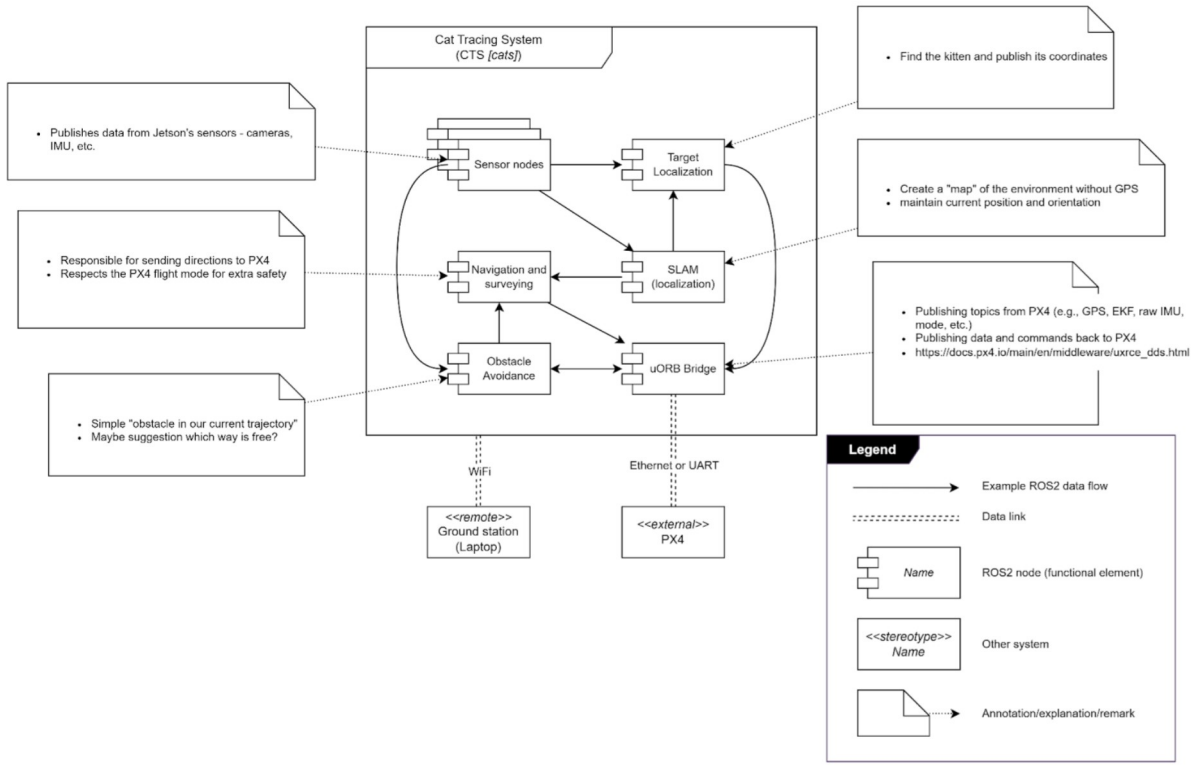
Information View

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

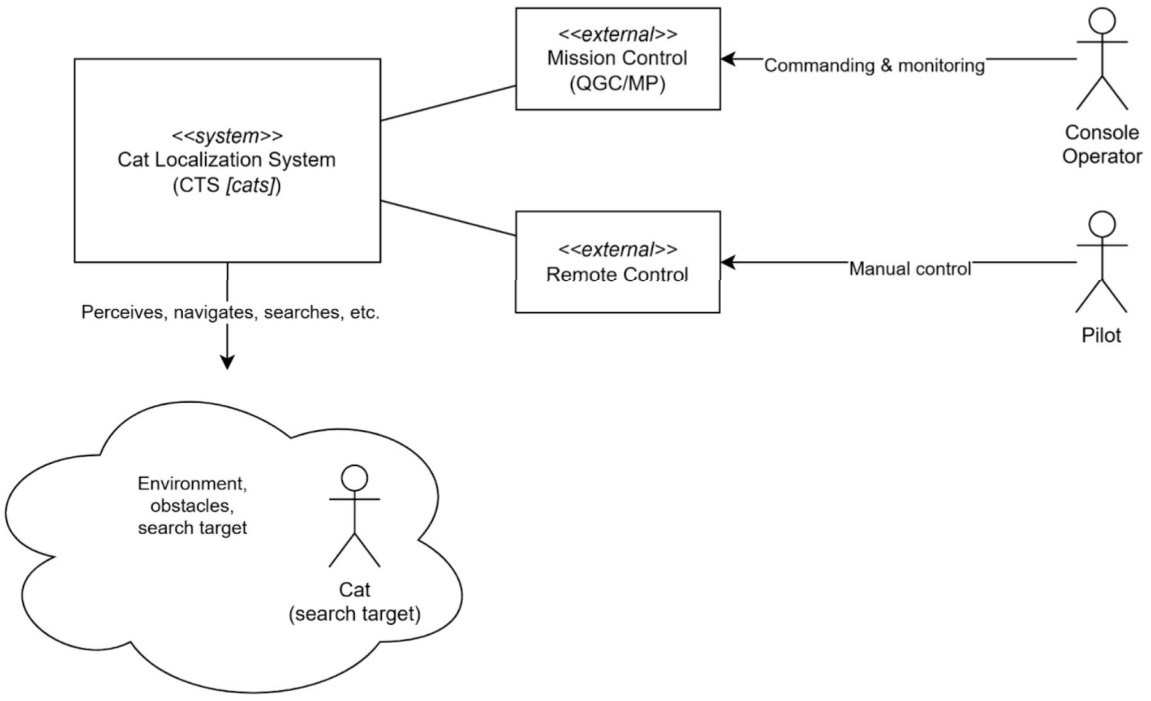
Sensor data acquisition and processing run concurrently with navigation computations to optimize performance and responsiveness. Simultaneously, communication with both the Pixhawk and the Ground Control Station (GCS) occurs in parallel to maintain real-time operational capabilities. ROS2's multi-threaded 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.

Functional View



As can be seen from the diagram above the system consists of six main components. Their responsibilities are shown in the notes near them in the diagram.

Context view



This diagram provides a high-level overview of the system in operation. Two people are responsible for ensuring that the system functions as expected. One person monitors the drone's state, ensuring, for example, that the battery level is sufficient and keeping track of the flight mode. The pilot is responsible for controlling the drone when necessary and reacting to any system issues. The drone should always respond to the pilot's commands, even if the programmed code instructs it to take a different action.

The goal is to design a drone capable of autonomously avoiding obstacles, analyzing camera feeds, and localizing a target object (a cat in this case) without relying on RTK or GNSS. Instead, localization is achieved using cameras and inertial measurement units (IMUs).