Find my kitten, AI!

A practical study in autonomous drone development

Kaius Koivisto, Ki Chun Tong, Konsta Mikkola, Martin Roznovjak, Imran Mamin, Mitja Rislakki, Olli Glorioso, Younes Elberkennou

Aalto University, School of Science

`github.com/CatScanners | catscanners.github.io/find-my-kitten`

Introduction

Why? Drones have the potential to revolutionize search and rescue operations by gathering critical information and navigating hazardous environments.

What? The goal of this project was a drone capable of autonomous navigation and the location, recognition, identification and moving near objects and communicating the findings to a control center. For safety and compliance, the system would always include a manual override option.

For whom? The project outcome (the product), its demonstration, and its feasibility study, is made to be used by researchers, investors or other developers for further assessement and development.

Main Objectives

1. Construct a drone ”package”: drone frame, companion computer, autopilot, real-time kinematic (RTK) positioning.
2. Construct a local development environment with Gazebo simulator and Robot Operating System 2 (ROS2).
3. Develop a ROS2 package for object detection and identification.
4. Develop a ROS2 package for maneuvering the drone and centering the drone on top of the recognized object.
5. Implement obstacle avoidance.
6. Create a Docker container for a SITL simulator setup.
7. Make the software work on the physical drone.
8. Document the work for reference and reproducibility.

Technologies

1. Software technologies: ROS2, Isaac ROS, PX4, Linux, NVIDIA JetPack, QGroundControl, Python, C++, Docker, Bash, OpenCV, PyTorch, Yolo v5, NumPy, etc.
2. Hardware technologies: NVIDIA Jetson Orin Nano and NX, Pixhawk 6X, IMX219 cameras, AR0234 Global Shutter Camera, Holybro Pixhawk Jetson Baseboard and Developer Kits, Holybro X500 Drone Frame, ZED Stereo Camera, OAK-D Camera, Radiomaster TX16S, ZED-F9P RTK GNSS, etc.

Technical Details

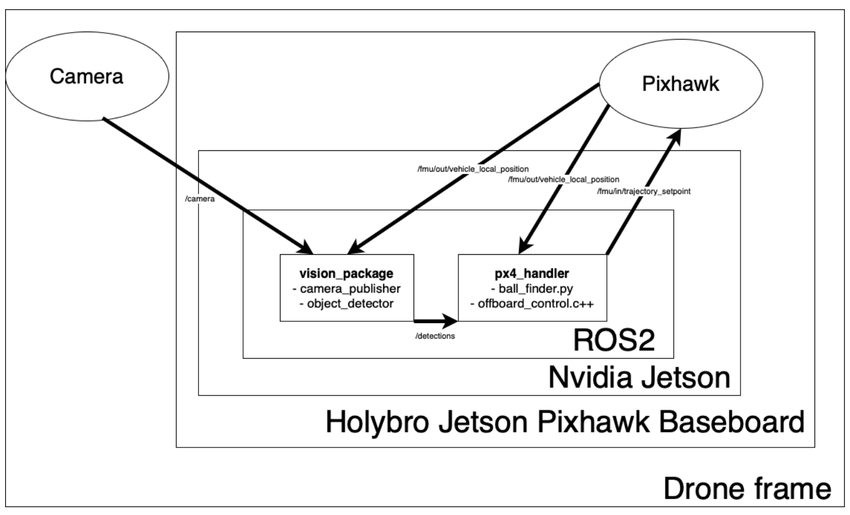

Figure 1: System architecture.

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.

Hardware setup

Displayed in Figure 4, The hardware used reflects the communication needs of the drone: The drone is controlled by a flight controller (Pixhawk), which can receive commands from an offboard computer (Jetson). A Ground Control Station (GCS) laptop is connected to the flight controller via telemetry radio for traditional flight control, other laptops are connected to the vehicle via a mobile hotspot to run and monitor scripts running on the offboard computer. A relay program provides virtual NTRIP (RTK) corrections source for RTK GPS. The drone is linked to a manual remote control radio for manual flight capabilities.

Results

1. A variety of tehcnologies and libraries were audited and applied. These ranged from robotics middleware (ROS2) to object recognition (YOLOV5).
2. Constructed a drone ”package” with a frame, companion computer, Pixhawk autopilot, and RTK GPS capabilities (Figure 2).
3. Constructed a local development environment with Gazebo + ROS2 + simulated camera,
4. Created a ROS2 package for object detection and identification.
5. Created a ROS2 package for maneuvering the drone and centering it on top of the recognized object. (Figure 3).

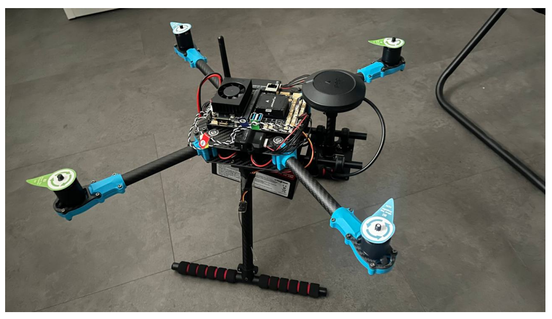

Figure 2: Drone frame with flight computers.

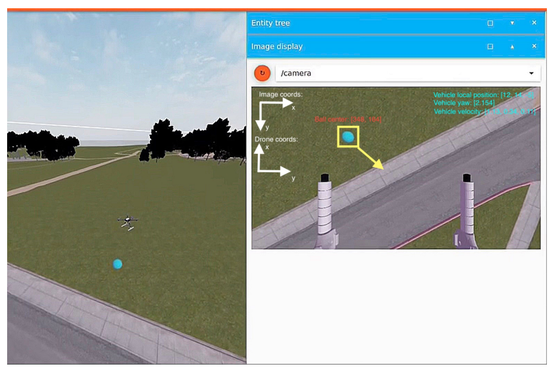

Figure 3: Simulator setup. (*Check text for illustration*)

6. We conducted 7 test flight sessions (+1 planned), a typical flight session took around 5-8 hours with at least 5 people always involved in the process. The first two flights were for testing safety procedures and basic drone functionality, as well as to test recording with the onboard cameras. Flights three and four were used to gather data for future obstacle detection. Basic offboard maneuvering was tested on flight five, and an improved version was tested on flight six. During fight seven, object detection and tracking was tested for the first time. Object detection proved to work well, but the tracking algorithm did not center the drone properly. The following features of the drone have been successfully implemented and tested in one of our flights (Figures 5, 6, 7):

- RTK positioning + QGroundControl motions.
- Two cameras (global shutter + ZED) worked and recorded footage successfully.
- Offboard control movements and surveying worked well.
- Object detection and logging worked well.
- Drone centering tested in simulator, test flight planned.

7. The project was comprehensively documented, and instructions for reproducing the current setup were written, along with Dockerfiles to make the process easier. These, along with the software that was developed, has been made available as a public GitHub repository: `catscanners.github.io/find-my-kitten`

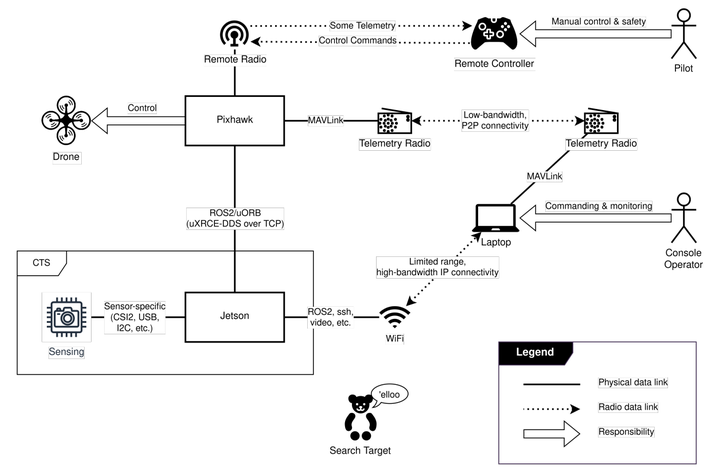

Figure 4: Hardware architecture.

Figure 5: Drone sunset.

Figure 6: Flight ground crew.

Figure 7: Downward camera.

Future development

The drone package and the simulator setup along with its extensive documentation now provides a good basis for future work that builds on this project. thanks to the simulator and development pipeline, feature development progressed significantly faster compared at the end of the project compared to the beginning. Some logical next steps for future development would be to implement obstacle avoidance with the already-working ZED camera, mapping and SLAM with the ZED camera and already-working Isaac ROS setup, and to make the solution work in low-light conditions.