ABSTRACT

Title of Thesis: INDOOR TARGET SEARCH, DETECTION, AND INSPECTION WITH AN AUTONOMOUS DRONE

Ahmed Ashry
Master of Science, 2024

Thesis Directed by: Professor Derek Paley
Department of Aerospace Engineering

This thesis investigates the deployment of unmanned aerial vehicles (UAVs) in indoor search and rescue (SAR) operations, focusing on enhancing autonomy through the development and integration of advanced technological solutions. The research addresses challenges related to autonomous navigation and target inspection in indoor environments. A key contribution is the development of an autonomous inspection routine that allows UAVs to navigate to and meticulously inspect targets identified by fiducial markers, replacing manual piloted inspection. To enhance the system's target recognition, a custom-trained object detection model identifies critical markers on targets, operating in real-time on the UAV's onboard computer. Additionally, the thesis introduces a comprehensive mission framework that manages transitions between coverage and inspection phases, experimentally validated using a quadrotor equipped with onboard sensing and computing across various scenarios. The research also explores integration and critical analysis of state-of-the-art path planning algorithms, enhancing UAV autonomy in cluttered settings. This is supported by evaluations conducted through software-in-the-loop simulations, setting the stage for future real-world applications.

INDOOR TARGET SEARCH, DETECTION, AND INSPECTION WITH AN AUTONOMOUS DRONE

by

Ahmed Ashry

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2024

Advisory Committee:
Professor Derek Paley, Chair/Advisor
Professor Michael Otte
Professor Joseph Conroy

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ABT	Adaptive Belief Tree
CPP	Coverage Path Planning
DDPG	Deep Deterministic Policy Gradient
EDF	Euclidean Distance Field
EKF	Extended Kalman Filter
ESCs	Electronic Speed Controllers
ESDF	Euclidean Signed Distance Field
GPS	Global Positioning System
GSC	Ground Control Station
HITL	Hardware in the Loop
IBVS	Image-Based Visual Servoing
IMU	Inertial Measurement Unit
IoU	Intersection over Union
LiDAR	Light Detection and Ranging
mAP	mean Average Precision
MAVLink	Micro Air Vehicle Link
MIQP	Mixed Integer Quadratic Program
MTOW	Maximum Take-Off Weight
NIST	National Institute of Standards and Technology
POMDP	Partially Observable Markov Decision Process
ROS	Robot Operating System
SAR	Search And Rescue
SITL	Software in the Loop
SLAM	Simultaneous Localization and Mapping
ToF	Time of Flight
UAVs	Unmanned Aerial Vehicles
VIO	Visual-Inertial Odometry
YOLO	You Only Look Once

Chapter 1: Introduction

1.1 Motivation and Objectives

In recent years, the world has witnessed an escalation in the frequency and intensity of natural and man-made disasters. The 2023 Turkey-Syria earthquake, for instance, resulted in catastrophic losses, underscoring the urgent need for effective disaster response strategies [1]. These incidents often lead to challenging scenarios where victims are trapped in inaccessible areas, necessitating rapid and efficient Search and Rescue (SAR) operations. Conventional methods, which rely heavily on human responders, are often hampered by physical risks, limited accessibility, and time constraints. This is where the deployment of autonomous small Unmanned Aerial Vehicles (UAVs), characterized by a maximum take-off weight (MTOW) of 13.5 kg or less [2], holds transformative potential. UAVs, equipped with advanced sensors and cameras, can access hard-to-reach areas, providing critical situational awareness without putting human lives at risk.

UAVs, commonly known as drones, have emerged as revolutionary tools in various domains, notably in disaster management and SAR. UAVs are instrumental in various post-disaster functions, including aerial damage assessment, victim localization, SAR coordination, and delivery of essential supplies [3]. They have been effectively utilized for quick evaluations of structural damage following earthquakes [4] and for innovative applications like customized defibrillator

payloads [5] and remote delivery of first aid kits [6]. Recent studies highlight UAVs' capabilities in rapid victim identification and condition assessment, with developments such as vision-based algorithms for detecting life signs [7] and computer vision techniques for victim detection [8].

However, the utilization of UAVs in SAR missions introduces a unique spectrum of challenges, particularly when these operations are conducted in indoor environments. The complexity of SAR operations in indoor environments is due to the constrained spaces, limited visibility, and complex interior layouts, all of which intensify the difficulty of locating and rescuing individuals [9]. In GPS-denied settings like collapsed buildings or dense urban areas, the conventional navigation systems in UAVs are rendered ineffective [10]. This calls for innovative solutions that can autonomously navigate, identify, and assess situations in such challenging environments. Autonomous drones can significantly expedite search operations, quickly locating victims and assessing structural damages, thereby guiding rescue efforts more effectively.

In summary, the motivation behind this thesis is driven by the need to innovate in disaster management and SAR operations using UAV technology. By developing frameworks and integrating algorithms for autonomous navigation, target detection, and inspection in complex indoor environments, this research aims to enhance the effectiveness, speed, and safety of disaster response operations, ultimately contributing to saving lives and mitigating disaster impacts.

1.2 Relation to Previous and Ongoing Work

The increasing exploration of UAV technology in the context of disaster management and SAR operations reflects a growing body of research dedicated to understanding and enhancing UAV capabilities and applications. This thesis is a trial to add to to this dynamic research field.

A significant portion of existing research focuses on the application of UAVs in outdoor environments, particularly in wide-area search missions following natural disasters. Studies highlight UAVs' ability to cover large areas quickly, making them invaluable in initial disaster assessment and victim location efforts [11]. This is complemented by research that delves into the integration of deep learning and computer vision for effective disaster management using UAVs. For instance, work done by Zhang et al. [12] discusses the development of advanced victim detection models tailored for UAV-based SAR operations.

However, a few recent studies focused on enhancing autonomy and efficiency in SAR operations within GPS-denied and cluttered environments. Sandino et al. (2020) discusses the development of an autonomous navigation system for small UAVs [13]. The core objective is to equip small UAVs with the ability to autonomously detect, localize, and quantify victims in disaster scenarios, leveraging a Partially Observable Markov Decision Process (POMDP) solved with an Adaptive Belief Tree (ABT) algorithm. Their approach integrates vision-based camera inputs to manage target detection uncertainties.

The methodology encompasses rigorous testing in simulated environments using Software in the Loop (SITL) with tools such as Gazebo, ROS, and PX4 firmware, supplemented by a Hardware in the Loop (HITL) setup. Despite its advancements, the study acknowledges certain limitations. The autonomous decision-making under environmental uncertainty and the modeling of target detection uncertainties from vision-based sensors remain complex challenges. Additionally, executing computationally intensive decision-making and object detection algorithms on resource-constrained hardware requires further optimization.

In their follow-up study [14], the focus shifts toward enhancing the UAV's operational autonomy in GPS-denied environments. The proposed framework incorporates an onboard com-

puting system, enabling the UAV to process data in real-time for effective navigation and target detection. The research demonstrates the application POMDP/ABT algorithm again, facilitating real-time decision-making and path planning for the UAV. Through rigorous testing, including HITL simulations and actual flight trials, the UAV showcased its ability to autonomously navigate and detect targets with varying degrees of uncertainty regarding their locations.

However, the study acknowledges certain limitations, particularly the reliance on pre-configured occupancy maps, which restricts the UAV's adaptability to dynamic environmental changes. The paper suggests future enhancements, such as real-time occupancy map updates and the inclusion of more sophisticated sensors, to further elevate the UAV's autonomous navigation and detection capabilities in complex indoor environments.

Pioneering studies such as that of Sampedro et al. (2019) presents a fully-autonomous aerial robot designed for complex SAR missions in unstructured indoor environments [15]. Their work focuses on integrating learning-based capabilities for target recognition and interaction, employing supervised learning classifiers and novel Image-Based Visual Servoing (IBVS) algorithms, hinged on deep reinforcement learning techniques like Deep Deterministic Policy Gradients (DDPG). The extensive validation across simulated and real-world settings underscores the UAV's adeptness in navigating complex, obstacle-laden terrains, effectively identifying targets, and engaging in precise interactions

1.3 Technical Approach

This research presents the development of an autonomous drone system designed for indoor exploration, target identification, and inspection. The primary objective is to automate the manual

processes of searching for, localizing, and inspecting targets within indoor environments, potentially applicable to SAR missions, disaster management, and exploration operations.

The system utilizes an algorithmic framework for indoor autonomous exploration. This framework integrates path planning and coverage algorithms, allowing the drone to systematically navigate and scan indoor environments. The approach employs a coverage path planning (CPP) strategy, specifically the boustrophedon or lawnmower pattern, ensuring complete coverage of the search area.

Upon detecting potential targets, the system focuses on accurate localization. This is achieved using a combination of fiducial markers (AprilTags) and advanced computer vision techniques. The detected markers aid in precise positioning and orientation assessment of the targets relative to the drone. This component of the system is critical for the detailed and accurate localization of the targets of interest.

Once a target is localized, the drone initiates an autonomous inspection protocol, which aims to examine the details of the targets using onboard cameras. This phase integrates an object detection model for identifying and classifying anomalies or objects of interest. The model, trained on a self-curated dataset of vision acuity markers, provides real-time analysis while running onboard on the drone, which improves the decision-making capabilities.

The project then advances to enhances the drone's autonomy in cluttered environments. This is addressed through simulation-based studies, implementing state-of-the-art path planning algorithms. The focus is on adapting the drone's navigation and operational strategies to more complex and realistic scenarios mimicing SAR operations.

1.4 Contributions

The primary contributions of this thesis are as follows:

1. **Autonomous Inspection Routine Development**: An autonomous inspection routine is developed and implemented, enabling UAVs to navigate to and inspect targets identified by fiducial markers. This routine replaces the manual inspection process, equipping the UAVs with the capability to autonomously perform comprehensive examinations of targets from multiple positions.

2. **Integration of Custom-Trained Object Detection**: A custom-trained object detection neural network is integrated to recognize vision acuity markers on targets. This model, trained on a self-curated dataset, is optimized, and deployed to the UAV's onboard computer, enabling real-time processing and augmenting the autonomous inspection routine's efficiency.

3. **Development of an Autonomous Mission Framework and Experimental Validation**: A comprehensive autonomous mission framework is designed to manage the transition between search and inspection phases effectively. It enables the UAV to autonomously explore areas, identify targets, and conduct in-depth inspections using the integrated object detection model for data acquisition before resuming exploration. The system is empirically tested in multiple autonomous mission scenarios.

4. **Adaptation and Analysis of State-of-the-Art Path Planning for Cluttered Environment Autonomy**: Integrated an open-source, state-of-the-art path planning algorithm into a software-in-the-loop simulation environment, detailing its implementation and offering critical analysis of its performance in different simulation environments.

1.5 Outline of Thesis

Chapter 2 provides a comprehensive background on UAV dynamics, control systems, the hardware used in experiments, and the software infrastructures that support autonomous flight. In Chapter 3, it details the development of autonomous exploration and visual inspection algorithms, integrating these systems for efficient target detection and localization. Chapter 4 explores the adaptation of these systems for navigation in cluttered environments, analyzing the performance of state-of-the-art path planning algorithms, and adapting one of them to our platform. Chapter 5 evaluates the effectiveness of the proposed systems through both simulated and real-world testing, presenting quantitative and qualitative results. Finally, Chapter 6 concludes with a summary of the research contributions and suggests directions for future work to enhance UAV autonomy further in complex environments

Chapter 2: Background

This chapter provides a comprehensive background essential for understanding the core technologies and methodologies employed in the thesis.

2.1 Indoor Navigation and Localization

2.1.1 Navigating GPS-Denied Environments

Indoor navigation for drones presents challenges, particularly in GPS-denied environments, which are paramount in SAR missions. GPS, where primarily used for outdoor navigation, becomes unreliable or entirely unusable in indoor, underground, or dense urban landscapes. This poses a significant problem for SAR missions, where timely and precise navigation is critical in environments like collapsed buildings or caves. In such scenarios, rescuers cannot rely on traditional GPS signals for localization, necessitating alternative methods for autonomous drones to navigate and fulfill their mission effectively. These environments are not only GPS-denied but often present complex, dynamic, and unstructured terrains that amplify the challenges for autonomous navigation [16].

2.1.2 State Estimation

To address these challenges, various techniques have been developed. Key among these is Visual-Inertial Odometry (VIO), a method that combines visual data (from cameras) with inertial measurements (from accelerometers and gyroscopes) to estimate the drone's position and orientation over time [17]. VIO operates by detecting visual features in the environment, tracking their motion over successive camera frames, and fusing this information with inertial data. This method is particularly effective in environments lacking GPS but rich in visual features. The advantages of VIO are its relative simplicity and the ubiquity of cameras and inertial sensors, which makes it a cost-effective solution for indoor navigation.

Another widely used technique in indoor state estimation is the Extended Kalman Filter (EKF). EKF is a statistical approach that continually estimates the state of a dynamic system (like the drone) from a series of incomplete and noisy measurements. In the context of drone navigation, EKF can be used to fuse various sensor data – from accelerometers, gyroscopes, magnetometers, and barometers – to provide a comprehensive and accurate estimate of the drone's position and motion [18]. This method is beneficial in environments where each individual sensor might not provide reliable data due to various environmental factors.

2.1.3 Fiducial Markers for Precise Indoor Localization

For enhancing localization accuracy, Fiducial Markers, such as AprilTags, are employed. AprilTags are simple, yet highly distinctive visual markers that can be easily detected and decoded by computer vision algorithms [19]. Once an AprilTag is detected, its known dimensions and the relative position of the camera allow for the precise calculation of the distance and orientation

of the tag relative to the drone. This is particularly useful for fine-grained positioning tasks required in SAR missions, where the drone might need to navigate to specific points or closely inspect certain objects or areas. AprilTags' simplicity and robustness against varying lighting conditions and viewing angles make them ideal for indoor use, where environmental conditions can be unpredictable.

2.2 Dynamics and Control of a Quadrotor Unmanned Aerial Vehicle

2.2.1 The Quadcopter Model

Discussing the dynamics of a quadcopter is essential as it's the key to understanding flight behavior. Modeling of such dynamics allows for the analysis and simulation of how a quadcopter moves through the air and responds to external inputs. This is the basis for the development of navigation and guidance algorithms for autonomous systems, and also for designing the control algorithms. Detailed analysis of such models can be found in [20, 21].

In order to specify the attitude, two frames of reference are defined as the inertial (e_I) and body frames (e_b). Fig. 2.1 shows the orientation of both frames.

The Euler angles vector containing roll, pitch and yaw is denoted by $\Theta = [\phi \quad \theta \quad \psi]^T$ and the angular velocity vector in the body frame is described as $\Omega = [p \quad q \quad r]^T$. Euler angles' time derivative ($\dot{\Theta}$) are related to angular velocities vector (Ω) using the following formula:

$$\dot{\Theta} = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & s\phi c\theta \\ 0 & -s\phi & c\phi c\theta \end{bmatrix} \Omega \tag{2.1}$$

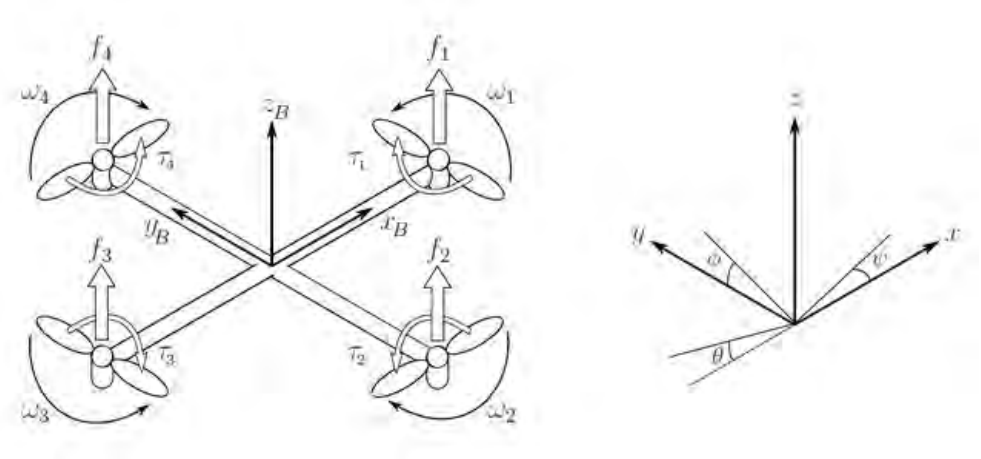


Figure 2.1: Quadcopter model with frames of reference [20].

The four rotors on the quadcopter are typically the source of generating the lifting force and moment. Under the assumption that the resulting forces and moments are proportional to the square of the rotor angular speed, the thrust and moment for each rotor are given by:

$$f_i = k_f \omega_i^2 \tag{2.2}$$

$$\tau_i = k_t \omega_i^2 \tag{2.3}$$

where f_i and τ_i are the force and moment generated from each rotor, k_f and k_t are the aerodynamic force and moment constants, and ω_i is the angular velocity of each rotor. The total thrust and torques as a result of the 4 motors working independently are expressed by:

$$\begin{aligned} T &= k_f[\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2] \\ \tau_\phi &= Lk_f[(\omega_2^2 + \omega_3^2) - (\omega_1^2 + \omega_4^2)] \\ \tau_\theta &= Lk_f[(\omega_1^2 + \omega_2^2) - (\omega_3^2 + \omega_4^2)] \\ \tau_\psi &= k_t[(\omega_1^2 + \omega_3^2) - (\omega_2^2 + \omega_4^2)] \end{aligned} \tag{2.4}$$

The translational equations of motion for the quadcopter in the inertial frame are given by:

$$m \begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix} \tag{2.5}$$

where **R** is the rotational matrix.

The rotational equations of motion are typically derived from the Newton-Euler method. It is assumed that the quadcopter is symmetrical, therefore the inertia matrix is diagonal. Rotational EOM for the quadcopter dynamics are defined as:

$$\begin{aligned} \ddot{\phi} &= \frac{\tau_\phi}{I_{xx}} + \frac{I_{yy}}{I_{xx}}\dot{\theta}\dot{\psi} - \frac{I_{zz}}{I_{xx}}\dot{\theta}\dot{\psi} \\ \ddot{\theta} &= \frac{\tau_\theta}{I_{yy}} + \frac{I_{zz}}{I_{yy}}\dot{\psi}\dot{\phi} - \frac{I_{xx}}{I_{yy}}\dot{\psi}\dot{\phi} \\ \ddot{\psi} &= \frac{\tau_\psi}{I_{zz}} + \frac{I_{xx}}{I_{zz}}\dot{\phi}\dot{\theta} - \frac{I_{yy}}{I_{zz}}\dot{\phi}\dot{\theta} \end{aligned} \tag{2.6}$$

After obtaining the complete equations of motions that describe the dynamics of the quadrotor, numerical integration methods are employed to solve the differential equations and simulate the quadrotor response over time. Since quadcopter dynamics are known to be highly nonlinear and strongly coupled, the system is linearized around an equilibrium point in order to simplify the mathematical equations and decouple the dynamics of the system.

2.2.2 Flight Control System

PX4 and Ardupilot are the two most prominent open-source flight controllers. They are used heavily in the field of aerial autonomous systems and are compatible with different aircraft

configurations including fixed-wing, multicopter, and VTOL aircraft. In this section, we will focus on the PX4 and its control architecture as it was used for the work in this thesis.

PX4 utilizes a standard cascaded control architecture for multicopter aircraft configuration employing a mix of P and PID controllers. Controllers are fed with state estimates that are generated from an Extended Kalman Filter. The overall cascaded flight controller takes an input of the position and yaw angle setpoints and calculates the required thrust setpoint for the rotors. The cascaded controller consists of four controllers which are; position controller, velocity controller, attitude controller, and angular rate controller. PX4 provides extensive documentation on each of the controllers mentioned and the methodology for tuning each of the controllers gains [22]. Fig.2.2 depicts the multicopter overall cascaded flight controller architecture used by PX4.

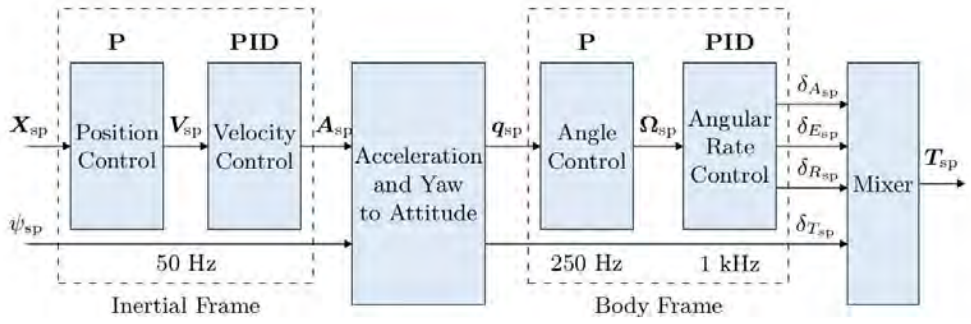


Figure 2.2: PX4 Multicopter Controller Architecture [22].

2.3 Software Infrastructure for Autonomous Drones

2.3.1 Robot Operating System

The Robot Operating System (ROS) is a software framework for developing robotic applications, highly used in autonomous drone research. It functions primarily as a middleware,

offering hardware abstraction, and message-passing functionalities. ROS's modular architecture facilitates independent development and testing of components, making it ideal for scalable system designs. This open-source platform encourages collaborative development and code reuse, significantly accelerating innovation in robotics.

ROS applications are typically composed of a number of nodes, each executing a specific function and communicating with other nodes over a messaging protocol. This modular architecture enables developers to build, test, and implement functionalities for their robotic platforms. The message-passing interface in ROS allows for asynchronous communication between nodes, which can be running concurrently on multiple machines. This system supports a variety of messaging types, including topic-based publish/subscribe messages and service-based request/response messages. The ROS Wiki [23] serves as the central documentation available online for ROS applications.

2.3.2 Communication with the Autopilot

Micro Air Vehicle Link (MAVLink[1]) is an open-source, lightweight messaging protocol, which is used for communication in drone systems. It facilitates efficient and robust data transmission between the drone and control stations. MAVLink's design ensures low overhead, which makes it highly efficient for real-time communication in resource-constrained environments.

MAVROS[2] represents an extension of this protocol within ROS. Essentially, MAVROS is a ROS package that enables seamless interaction between ROS-based programs and drone autopilots through the MAVLink protocol. It operates typically on a companion computer, inter-

[1]MAVLink documentation: https://mavlink.io/en
[2]MAVROS documnentaion: https://wiki.ros.org/mavros

facing with the Autopilot, like PX4, to execute autonomous commands. MAVROS facilitates this by sending specific control setpoints to the flight controller, with a safety timeout of 0.5 seconds. In setups utilizing PX4 autopilots, the offboard mode in PX4 facilitates this execution of setpoints sent over MAVROS messages.

2.3.3 Software in the Loop Simulation

Software-in-the-Loop (SITL) simulation is an important aspect of drone development since it allows for virtual testing of different algorithms and system functionality. It is crucial for evaluating flight control algorithms, sensor processing pipelines, and interaction with external systems without the risk and expense of real-world testing. SITL simulation offers the ability to thoroughly test and debug in a controlled, user-defined environment to ensure reliability and test outcomes before real-world deployment. Within PX4, SITL simulation involves running the flight stack software on a computer, replicating the vehicle's behavior in response to simulated sensor inputs and control commands. The PX4 framework uses the Simulator MAVLink API to connect with various simulators, which facilitates the exchange of information between the virtual environment and the flight control software [24].

In SITL, communication between PX4 and the simulation environment is achieved via MAVLink messages. Sensor data from the simulation, such as IMU readings and camera streams, are fed into PX4, while PX4 sends actuator control signals back to the simulator. This bi-directional communication ensures that the simulated drone reacts as it would in a real-world scenario. As shown in Fig. 2.3 shows the standard simulation components within PX4, which is agnostic to the type of simulator used. It includes PX4 on SITL, QGroundControl[3], and the

[3]QGroundControl is used for mission planning with MAVLink equipped drones

simulator, interconnected via UDP and TCP connections. PX4 connects to the simulator's local TCP port 4560 exchanging sensor and actuator messages to mirror real-time flight conditions. QGroundControl connects via remote UDP ports to provide a user interface for MAVLink telemetry messaging.

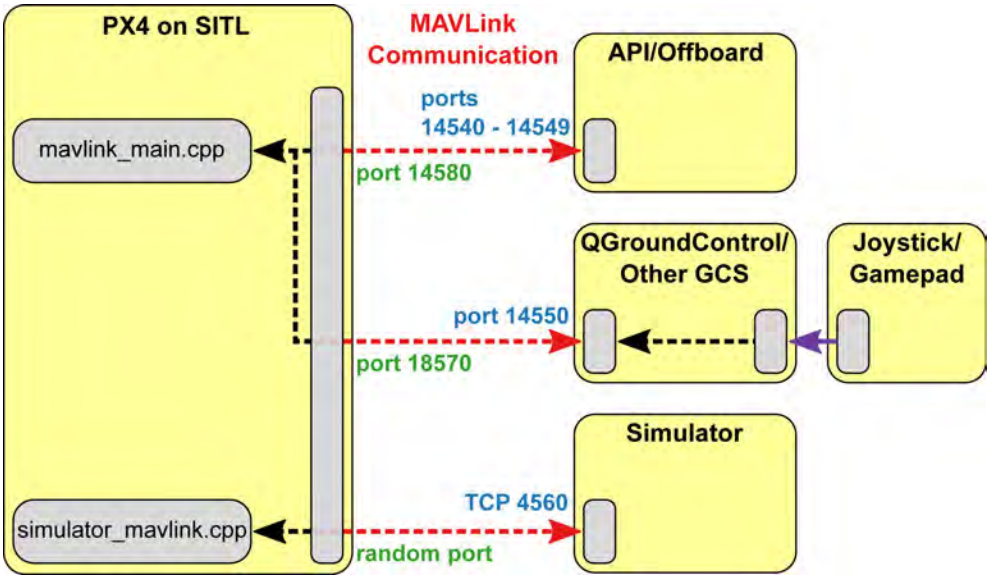


Figure 2.3: Standard SITL simulation environment within the PX4 stack. Adapted from [24]

Gazebo is a 3D simulation platform, which enables seamless simulation of diverse drone models running the PX4 flight stack, like Iris, and it allows for adding different sensors and cameras to the virtual drone [25]. Fig. 2.4 shows an example of the Iris drone, equipped with a downward facing camera, and spawned in an environment with a fiducial marker on the ground. Gazebo's compatibility with ROS topics allows for real-time data access and analysis, where Gazebo provides a dynamic virtual environment and sensor data, PX4 handles flight control, and ROS facilitates data handling and autonomy software testing.

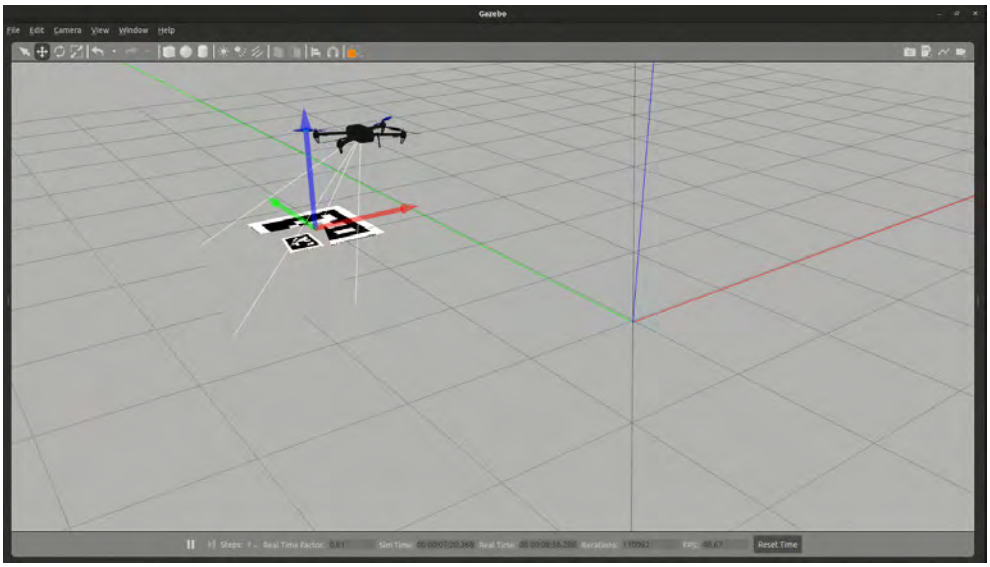


Figure 2.4: Snapshot from Gazebo SITL simulation with Iris quadcopter equipped with a downward facing camera and looking at a virtual fiducial marker

2.4 Experimental Testbed Overview

2.4.1 Drone Hardware Configuration

The M500 drone, by ModalAI, serves as a reference platform for autonomous flight research and development [26]. As shown in Fig. 2.5, the drone is equipped with 2216 880KV motors and 10-inch propellers, with a flight time of over 20 minutes and a payload capacity of up to 1 kilogram. The Electronic Speed Controllers (ESCs), rated at BLHeli 20A, are placed under each arm for power delivery and flight control. Structurally, the M500 has a 15.5-inch length and width, and a total take-off weight of 1075g including the battery. It hosts a suite of image sensors, including a 4K high-resolution camera for capturing detailed visual data, a stereo camera pair for depth perception, and a tracking camera to maintain orientation and positional awareness.

At the heart of the M500's operation is the VOXL Flight Deck, which merges the VOXL companion computer with the Flight Core flight controller. The VOXL companion computer,

Figure 2.5: m500 drone setup with key hardware components labeled

powered by a Qualcomm Snapdragon processor, brings powerful onboard computing to the drone, which is the main part for data processing. Meanwhile, the Flight Core, equipped with the PX4 flight control software, ensures precise flight dynamics and real-time responsiveness to control inputs.

2.4.2 Drone Software Functionality

The software ecosystem of the M500 drone is running on the VOXL companion computer, which comes with a Linux Yocto operating system. This comes with OpenCV and ROS Indigo installed. At the heart of VOXL lies the Qualcomm Snapdragon 821 processor, which offers robust computational power with its quad-core CPU and integrated GPU and DSP. This makes it suitable for vision-based applications and autonomous drone programs. For development on the companion computer, Docker is used to run Ubuntu 20.04 and host different ROS packages. This

containerization platform enables VOXL to run various operating systems and software stacks in isolated environments. Fig. 2.6 shows the system overview and the interconnectivity of the VOXL platform's software parts. Specifically, WiFi is used for communication in this project, which allows the communication between the drone and the Ground Control Station (GSC).

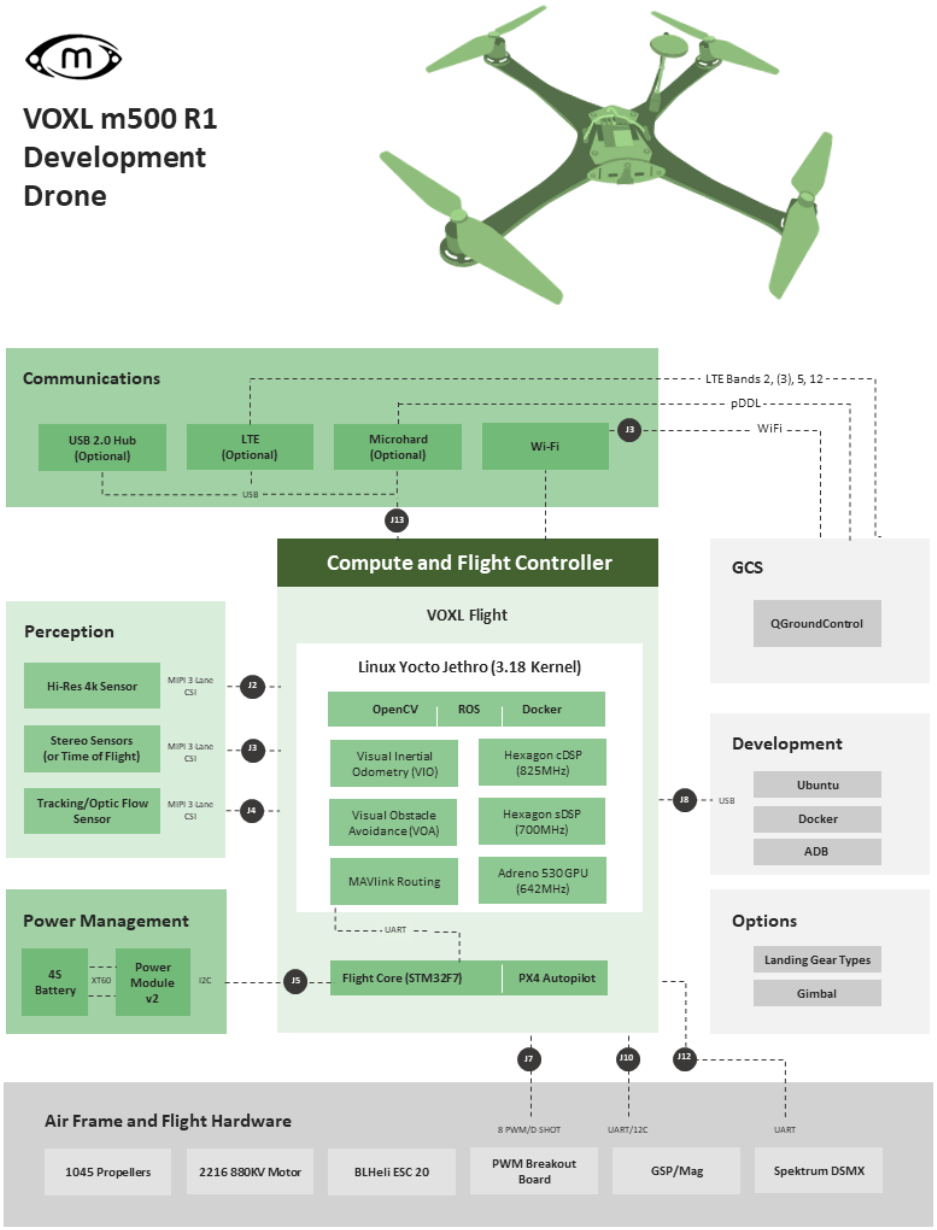


Figure 2.6: m500 VOXL platform software architecture

VIO is provided by the voxl-vision-px4 service, and it is used to navigate indoors where GPS signals are unavailable. PX4 interprets the VIO data and fuse them using EKF2 to provide the drone with its own state estimate while navigating indoors. This is then interfaced using MAVROS, which serves as a bridge for commands from higher-level algorithms, developed in ROS, to the PX4 flight controller.

2.4.3 Experimental Facility Setup

The Brin Family Aerial Robotics Lab at the Maryland Robotics Center features a 430 square-foot netted test area, shown in Fig. X, with 15-foot high ceilings suitable for indoors flight tests. The net is made with knotted nylon mesh, and the floor is lined with dual layers of foam mats for protection during the tests. The lab is equipped with Vicon[4] motion capture system, consisting of 12 Vantage V8 cameras. This provides precise tracking within the space, and is used as the source for ground truth data during experiments.

[4]Vicon motion capture system: `https://www.vicon.com/`

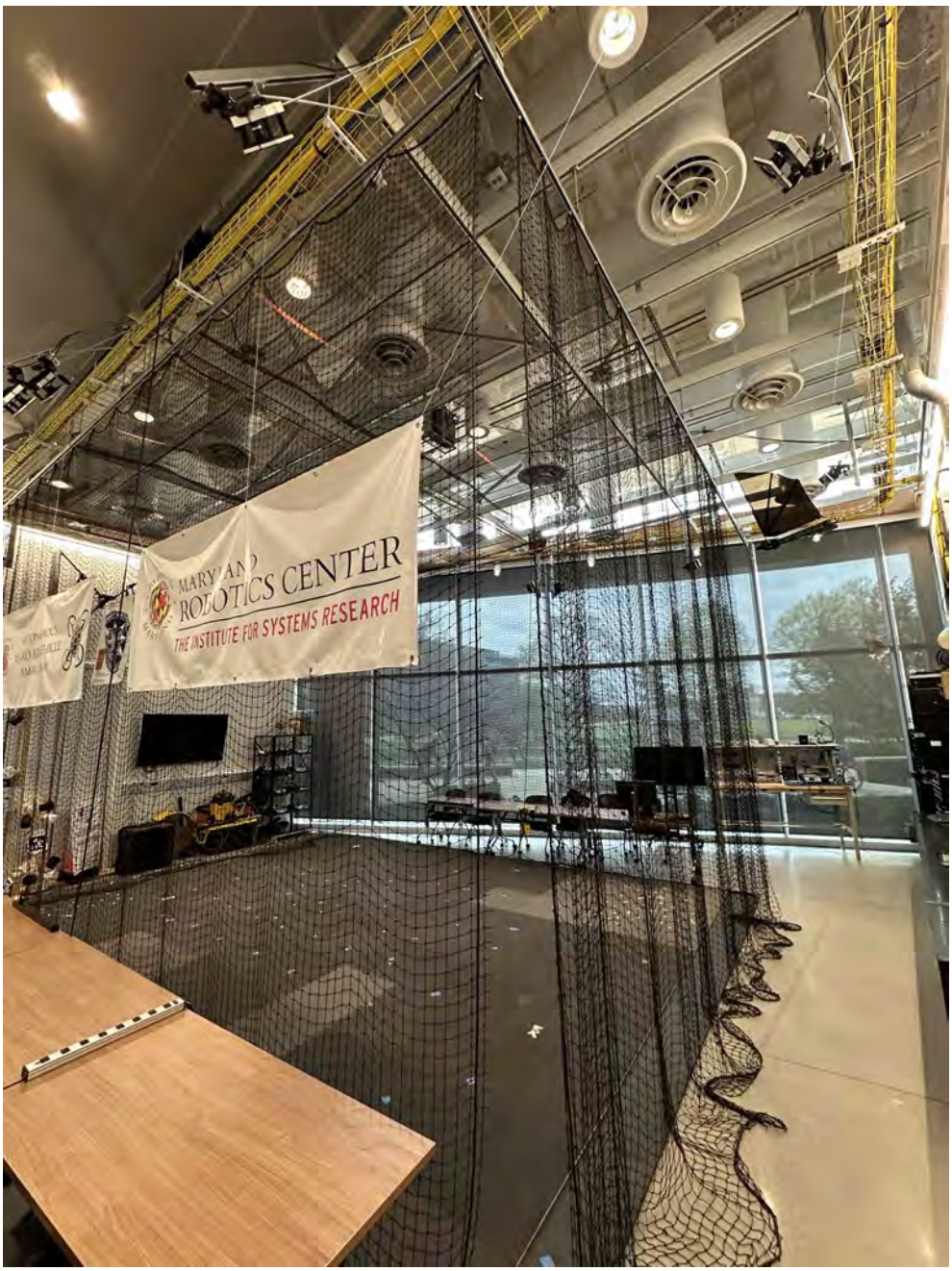


Figure 2.7: Netted Area in the Maryland Robotics Aerial Robotics Lab

Chapter 3: Autonomous Maneuvering and Visual Inspection

This chapter discusses different modules of the autonomous mission, along with the integration of those modules together. It starts with breaking down the search algorithm in section 3.1, then it shows the details of the chosen targets for this mission and the process of localizing those targets in section 3.2. After detecting and localizing a target of interest in the search area, the agent starts the inspection module, which is shown in section 3.3. During inspection, the system detects acuity vision markers on the target by running a custom-trained neural network, which is detailed in section 3.4. Finally, the process of integrating all the modules and dynamically switching between them is shown in section 3.5.

3.1 Autonomous Search Algorithm

3.1.1 Development of the Search Strategy

In the scope of motivation of this work, which is to support first responders in their SAR missions, the autonomous drone system's mission is to provide a reliable and efficient means to explore an indoor environment autonomously. This could be pictured as the operator initiating a search sequence with the actuation of a switch. The drone starts a predefined search pattern to scan the area of interest systematically while looking for targets. This application is rooted in

the area of CPP. CPP is a framework that contains methodological ways to ensure that a robot traverses all points within a predefined space, following an optimal trajectory [27].

Coverage of all the points while exploring an environment is a critical component in SAR operations. While various search patterns exist, which range from probabilistic approaches, such as random search, to exhaustive and systematic strategies like grid-based and boustrophedon searches. The boustrophedon, known as the lawnmower pattern, is named after the ox-driven plough's back-and-forth pattern, and is recognized for its systematic completeness [28]. It is particularly effective in environments with known dimensions and that are free of obstacles, which resemble the initial experimental facility at which we will be testing the implementation. The choice of this strategy over others is substantiated by its inherent ability to ensure complete coverage, which is a characteristic particularly advantageous in random target distribution scenarios. The lawnmower pattern's efficiency is further validated when considering indoor environments where uncertainty in localization and sensory inputs is a factor, with studies showing its superiority over random policies under such constraints [29].

Within a predefined indoor space, optimizing the search strategy requires careful consideration of the drone's operating altitude and pass width. This trade-off should be carefully made, considering how to minimize unnecessary overlaps without overlooking parts of the environment, while also having a sufficient footprint on the ground that suits the specific use case of the mission. To illustrate, the higher altitude you fly at, the more area you cover in less time, but that also means less resolution. Hence, this depends on the sensor capabilities of the camera being used, where the algorithm sets the drone's flight altitude to maximize the field of view while maintaining the ability to detect objects of interest.

3.1.2 Algorithm Implementation: Waypoint Generation and Path Planning

The first step of the autonomy mission is the algorithm that governs the generation and publication of the search waypoints. With the user-defined parameters of room length, room width, desired altitude, and resolutions along the X and Y axes, the algorithm systematically partitions the search area into a grid, generating a sequence of waypoints that guide the drone in a lawnmower pattern to cover the entire area as shown in Fig. 3.1. These resolutions dictate the granularity of the search grid, determining the drone's path precision in each axis's direction.

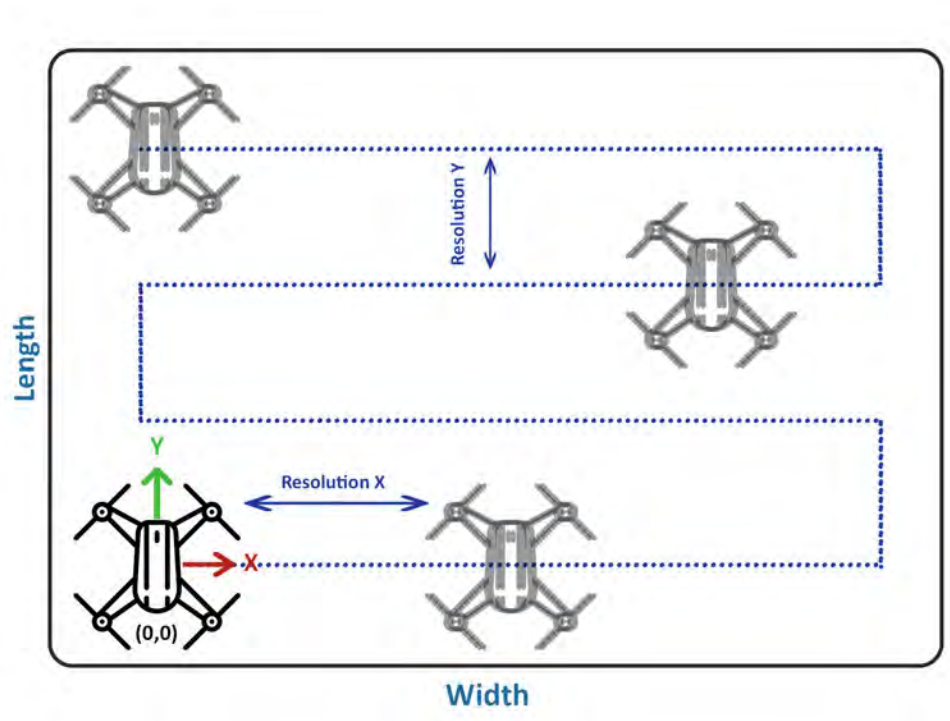


Figure 3.1: Lawnmower search pattern illustration

The desired height is adjustable to accommodate varying conditions and camera resolutions, influencing the coverage time and search efficiency. Given the specs of the camera [1] used in the experimentation of this work, and the size of the targets (printed AprilTag on US letter sizes

[1] VOXL M0014 tracking sensor

paper) that the drone is searching for, it has been empirically determined that it is detectable at a maximum altitude of 1.2 meters.

The process, as illustrated in the flowchart shown in Fig. 3.2, starts with generating the waypoints as a list of pose setpoints that contains the position and orientation of the drone at each step to cover the area. These points form are arranged in parallel lines with the drone changing direction at each room's end for complete coverage. Once generated, the waypoints are published to the drone with `MAVROS` as a `PositionTarget` message on the 'mavros/setpoint_raw/local' topic, which handles the translation from the ENU frame used in generating those waypoints to the NED frame utilized by PX4.

The drone starts at the home position (0,0) and maintains a fixed heading throughout the mission, as indicated by a 90-degree positive yaw. The algorithm ensures each waypoint is reached by comparing the drone's current pose, obtained from the state estimation module reading from 'mavros/local_position/pose' topic, with the target waypoint that is being published. If the drone is within a user-defined acceptance radius, it proceeds to the next waypoint. This ensures the drone adheres to the planned path by checking its position against the current waypoint and updating the waypoint index only when the position criteria are met. This process repeats until the drone has traversed the entire search area, after which it returns to the home position signaling the completion of the search pattern.

Fig. 3.3 shows the execution of the search mission in a SITL setup. As shown, a simulated drone has covered a predefined area of interest in Gazebo.

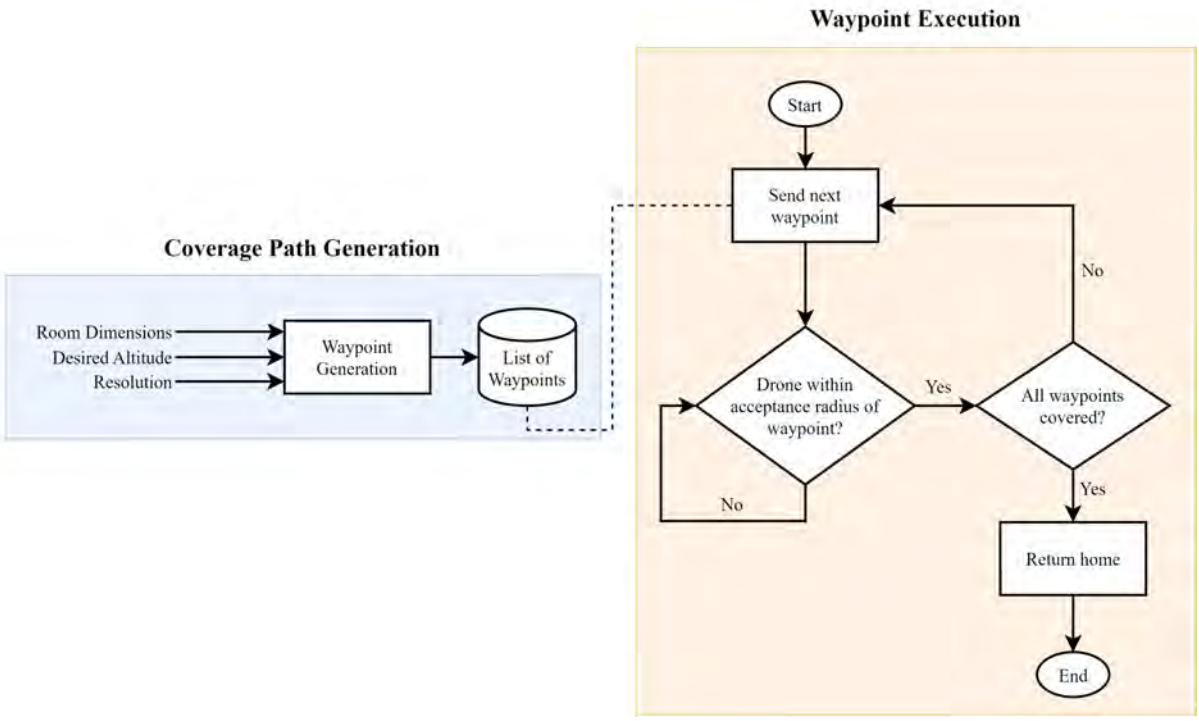


Figure 3.2: Search waypoints generation and execution flowchart

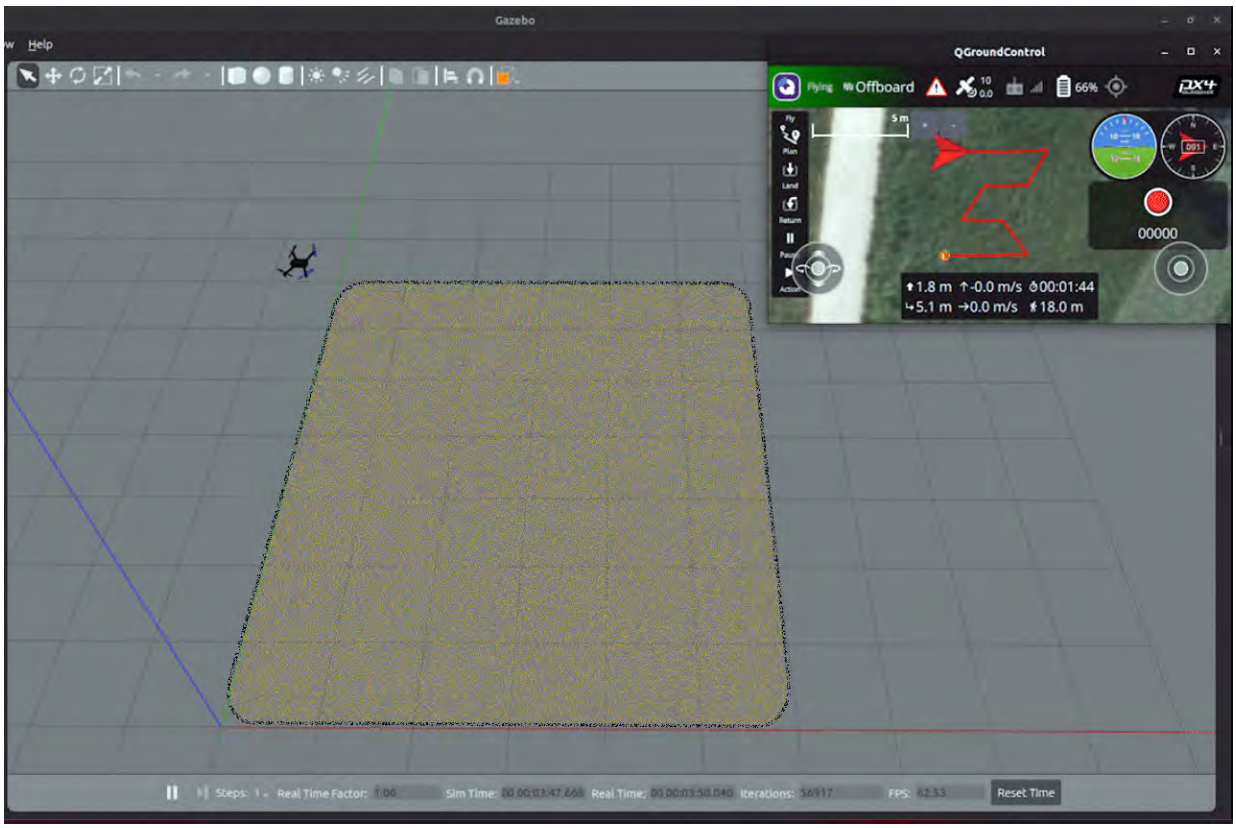


Figure 3.3: Search mission in simulation

3.2 Target Detection and Localization

3.2.1 Target Specifications and Rationale

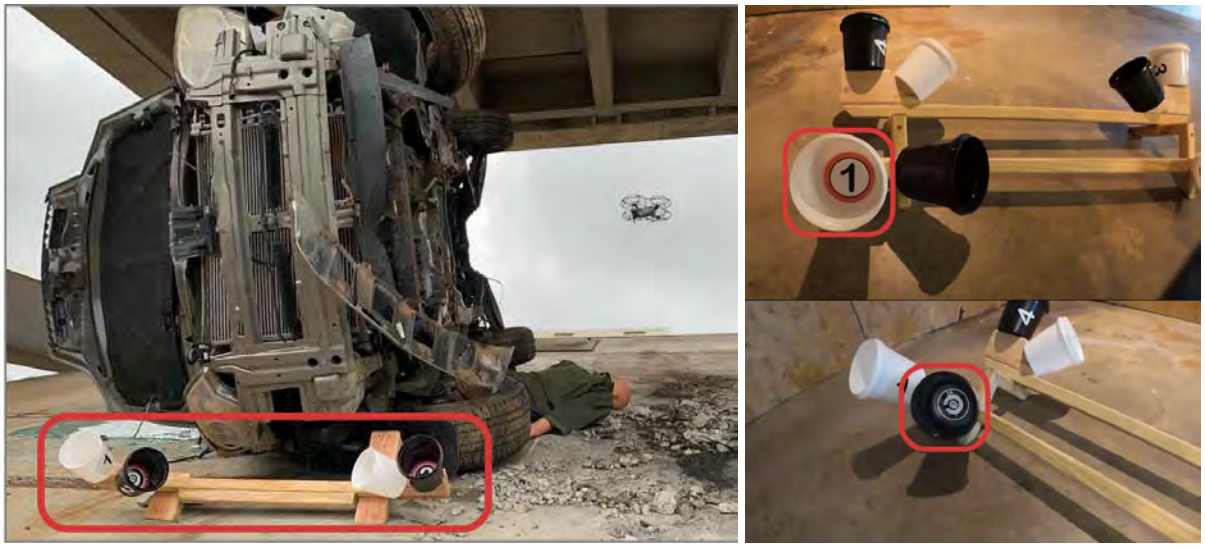
The goal of this research, as outlined in Chapter 1, is to enhance the efficiency and response speed of first responders in SAR missions. This involves not just locating human casualties but also inspecting them and performing triage assessments, as well as identify-ing environmental hazards. The ultimate goal is to increase the level of autonomy in these operations. This objective also guided our choice of experimental targets, which are bucket configurations as defined by the National Institute of Standards and Technology (NIST) First Responder UAS Indoor Challenge [2]

NIST crafted these targets to assess the localization capability and sensor acuity of UAV platforms. The targets consist of various bucket assemblies, designed to evaluate the operational readiness and capabilities of the system. These alignments, due to their replication ease, offer a robust framework for evaluation. They challenge the drone's ability to operate in confined spaces, dusty environments, and scenarios where targets are dispersed and oriented diversely. This setup tests the drone's onboard sensors in detecting and inspecting small labels placed inside the buckets.

NIST's intention with these targets is to establish a standardized method for testing and evaluating robotic platforms in emergency situations, a practice that is gaining traction in similar scenarios. The reasons mentioned above were pivotal in choosing these targets to evaluate our project's mission. Fig. 3.4a illustrates a bucket alignment used in a confined test scenario, depicting a collapsed environment unsafe for responders. The figure highlights the bucket align-

[2]NIST UAS Indoor Challenge: information about the contest

ment amidst crushed cars and semi-collapsed structures, showcasing the aerial vehicle's potential in such challenging environments.



(a) Bucket alignment in a disaster scenario (b) Acuity vision labels

Figure 3.4: Target alignment for testing

Fig. 3.4b focuses on a dual bucket alignment, where a perpendicular bucket pairs with an angled one, forming a right triangle. The buckets, each holding 1-liter (1-quart) capacity with 10 cm (4 in) diameter targets inside, are spaced 1 m (3 ft) apart. The alignment rings within the buckets challenge the acuity of the onboard sensors. These targets, often in black and white, test the UAV's response to various lighting conditions and its ability to closely inspect targets. The ease of replicating these acuity vision labels further justifies their use in testing our autonomous target localization and detection system.

Fig. 3.5 presents a model of one such bucket alignment, alongside an actual alignment used in our laboratory tests. There are various configurations, differing in bucket placement, label designs, and orientations, which makes it necessary that the autonomous agent not only localizes targets within an area but also identifies and distinguishes between different targets. This aspect

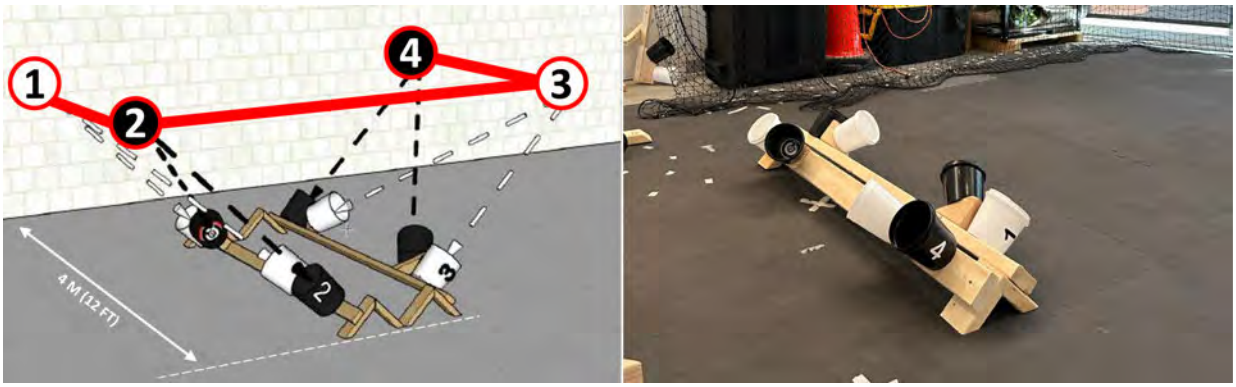


Figure 3.5: Model vs. actual configuration of target alignment - Left: a graphical representation of "Ground" bucket alignment with labels showing order. Right: the corresponding real-world laboratory setup used for UAV inspection trials.

will be explored in greater detail in the following subsection.

3.2.2 Fiducial Marker-Based Identification

To meet the goal of effectively identifying and pinpointing targets within the local inertial frame, our approach employs fiducial markers as a reliable reference. These markers are designed with distinct characteristics that facilitate their robust detection.

AprilTags, a type of visual fiducial marker, play a big role in robotic vision-based localization, and they serve as accurate landmarks for autonomous systems [19]. They consist of a set of black and white patterns, similar to QR codes, which are easily detectable, extractable, and trackable by computer vision systems. Utilizing such clear, structured patterns simplifies the recognition and tracking process in computer vision applications which are often subject to noise, varying lighting conditions, and computational complexity. The known dimensions of these tags address the issue of scale ambiguity, while their unique orientations enable the determination of their 3D pose relative to the camera. The diverse tag families, including the Tag `36h11` family used in this thesis, are illustrated in Fig. 3.6.

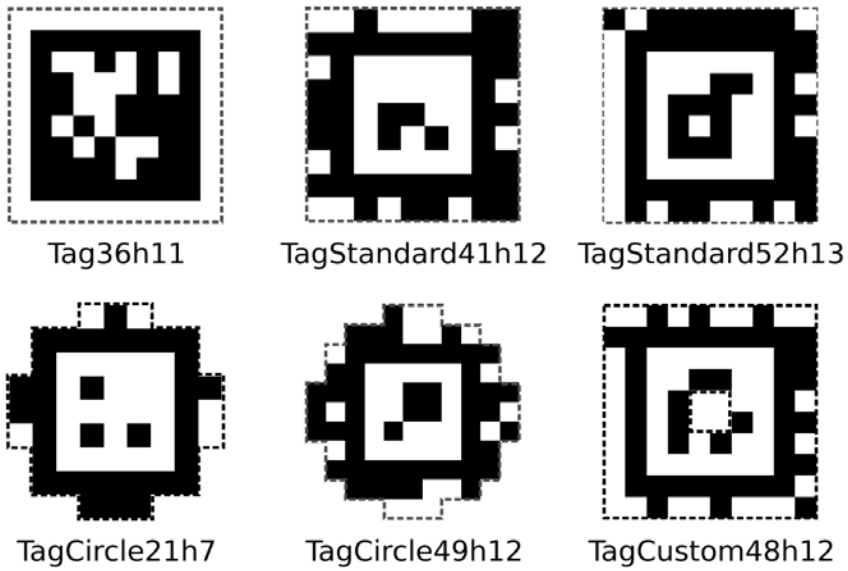


Figure 3.6: Different AprilTags families

In our experiments, AprilTags, each encoded with a unique ID, are attached to bucket assemblies to differentiate targets, as depicted in Fig. 3.7. This method allows the algorithm to recognize various targets during the search process. The figure demonstrates the three principal target alignments tested: Target 0 (T0), Target 1 (T1), and Target 2 (T2), as shown from left to right.

The operational methodology involves processing images from the onboard camera through the perception pipeline. Upon detecting an AprilTag, the algorithm computes a relative pose. This calculation enables the drone to localize the detected tag in relation to its own position and concurrently ascertain the specific target being observed.

3.2.3 Pose Estimation and Filtering

Once the drone detects an AprilTag from the downward facing camera, the tracking camera, it passes this as a tag detection to another module running onboard on the companion computer,

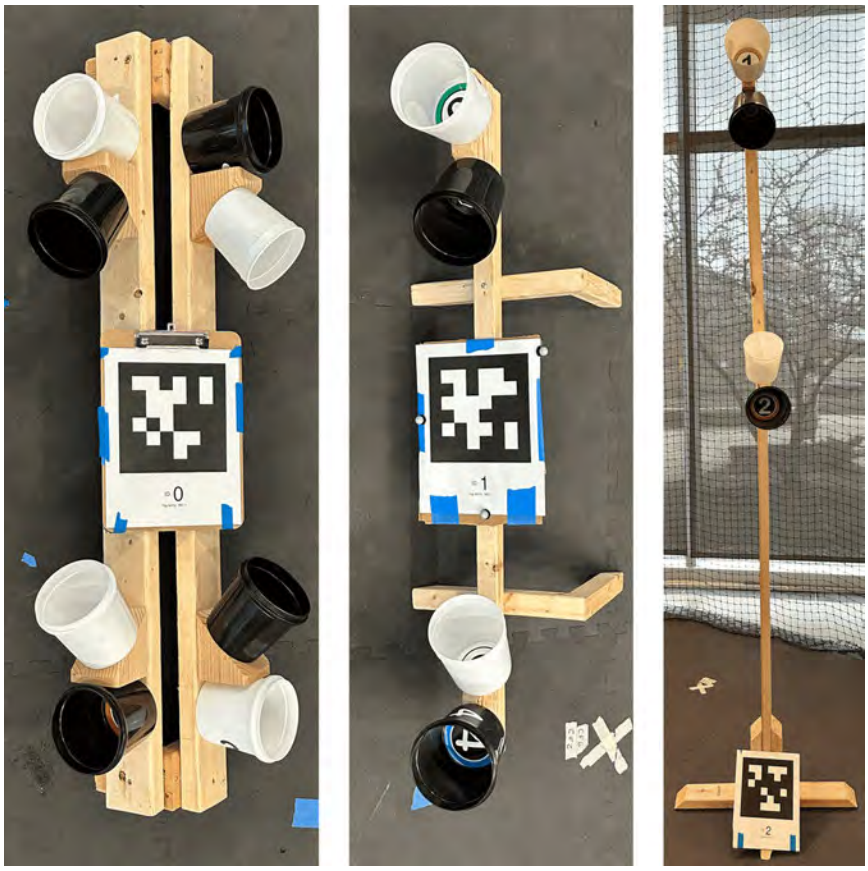


Figure 3.7: Multiple target identification with different tag IDs. From left to right, Target 0 (T0), Target 1 (T1), Target 2 (T2)

which then computes the tag_pose and then we can access this pose as a ROS message. This process is depicted in the block diagram shown in figure 3.8 These processes are running onboard on the companion computer in real time.

This project utilizes an established ROS-based AprilTag detection package, desig-nated as `apriltag_ros`[3]. This package implements the AprilTag3 algorithm, encapsulated within a ROS wrapper. It processes camera images, post rectification, and outputs both the detected tag ID and the tag's pose relative to the camera frame. Breaking down this process, the term 'rectified image' refers to an image corrected for distortion. This rectification necessitates a camera calibration process to determine the camera's intrinsic parameters, which are needed for

[3]apriltag_ros wiki page: https://github.com/AprilRobotics/apriltag_ros

the tag detector's effective operation.

Regarding the term 'pose', it refers to a 3D pose encompassing both the position and orientation of an object. Throughout the sections of this chapter, 3D poses are represented as homogeneous transformations. These are expressed as 4×4 matrices that combine both rotation and translation components. This matrix format allows for the comprehensive definition of an object's position and orientation in space. The standard configuration of this matrix is:

$$ {}^{A}T^{B} = \begin{bmatrix} R & \mathbf{p} \\ \mathbf{0} & 1 \end{bmatrix} $$

${}^{A}T^{B}$ represents a transformation matrix from frame B to frame A, where R is the rotation matrix and $\mathbf{p}$ is the position vector.

The block diagram in Fig. 3.8 outlines the process beginning with the raw image stream from the tracking camera. This stream is first subjected to an image rectification algorithm utilizing a pinhole camera model for calibration[4]. This calibration step is crucial to accurately estimate the tag's pose relative to the camera frame. Following this, the AprilTag detection ROS node, implementing the AprilTag 3 algorithm [30], processes the rectified images. It outputs the identified tag ID and its pose as a "geometry_msgs/PoseWithCovarianceStamped" message. The ROS wrapper's functioning is explicitly defined by two configuration files: config/tags.yaml (identifying specific tags and tag bundles for detection) and config/settings.yaml (configuring the core AprilTag 3 algorithm).

The system is efficiently adapted to operate within the VOXL companion computer's pipeline architecture. In this setup[5], the tag detection operates as a systemd service. It actively monitors

[4]Camera Calibration Framework

[5]AprilTag Detection on VOXL: `https://docs.modalai.com/voxl-tag-detector-0_9/`

the camera feed, identifying tag detections, which are subsequently converted to ROS format using 'voxl_mpa_to_ros'. For computational efficiency, the system is designed to bypass processing of five consecutive frames, focusing instead on every sixth frame. This frequency is adjustable to suit specific requirements, acknowledging that continuous processing of every frame is generally unnecessary and would excessively burden the CPU.

The primary objective remains to precisely locate the target, marked by an AprilTag, within the local inertial frame. This is accomplished by utilizing the ROS node output, which provides the pose of the detected AprilTag and publishes it on the ROS topic '/tag_detections/tagpose'. Effectively, by determining the AprilTag's location relative to the camera frame, we are concurrently localizing the bucket arrangement in the same frame. By also incorporating the drone's pose data in the local inertial frame from the rostopic '/mavros/local_position/pose', we can transpose the AprilTag's pose from the camera frame to the local inertial frame.

To detail the process:

- The drone's estimated pose in the inertial frame is denoted as the homogeneous transformation ${}^{I}T^{B}$.
- The estimation of the AprilTag's pose in the camera frame is represented as ${}^{C}T^{A}$.
- The static transformation between the drone's camera frame and body frame is noted as ${}^{B}T^{C}$, which could be acquired from the drone's CAD model. For the m500 drone, this transformation is determined.

With these transformation matrices at hand, we can employ the principles of rigid transformations to find the AprilTag's pose in the inertial frame. This is achieved by concatenating these transformations in the following sequence:

$$ {}^{I}T^{A} = {}^{I}T^{B}\ {}^{B}T^{C}\ {}^{C}T^{A} \tag{3.1} $$

While Equation 3.1 determines the AprilTag's pose in the local inertial frame, it's important to recognize the inherent noise and uncertainty present in sensor detections. To counteract these variances, the algorithm incorporates a filtering strategy to refine the pose estimates. For example, when an AprilTag is identified, the drone momentarily pauses and hovers to stabilize the pose data collection, thereby reducing noise and uncertainty in the sensory input. This precision in pose estimation is crucial for the drone's subsequent navigation and inspection tasks.

The algorithm enhances these pose estimations through a systematic process of transformations and verifications. Initially, the detected pose, given in 'geometry_msgs::PoseStamped', is transformed into a homogeneous matrix that aligns the AprilTag's pose with the camera frame. A key feature of the algorithm is its iterative filtering technique. It constantly compares new pose estimations against previous ones, accepting only those within specific positional and orientational thresholds. This approach ensures the consistency and reliability of the data. In situations where stable pose estimations are critical, the algorithm utilizes a 'bestPose()' function. This function engages outlier rejection and averages the translation and rotation elements from gathered data to deduce the most probable pose. This rigorous process results in a robust and reliable estimate of the AprilTag's pose in the inertial frame.

3.3 Autonomous Visual Inspection of Targets

The goal of this part of the mission is to command the drone autonomously to certain poses so that the drone's tracking camera can see the vision markers printed at the bottom of

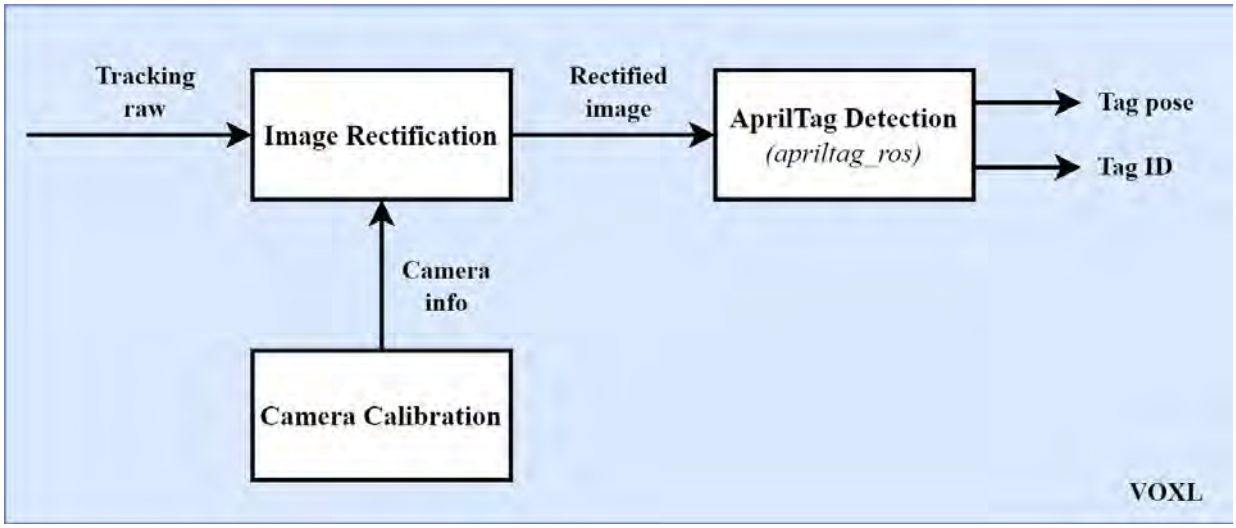


Figure 3.8: AprilTag detection running on VOXL block diagram

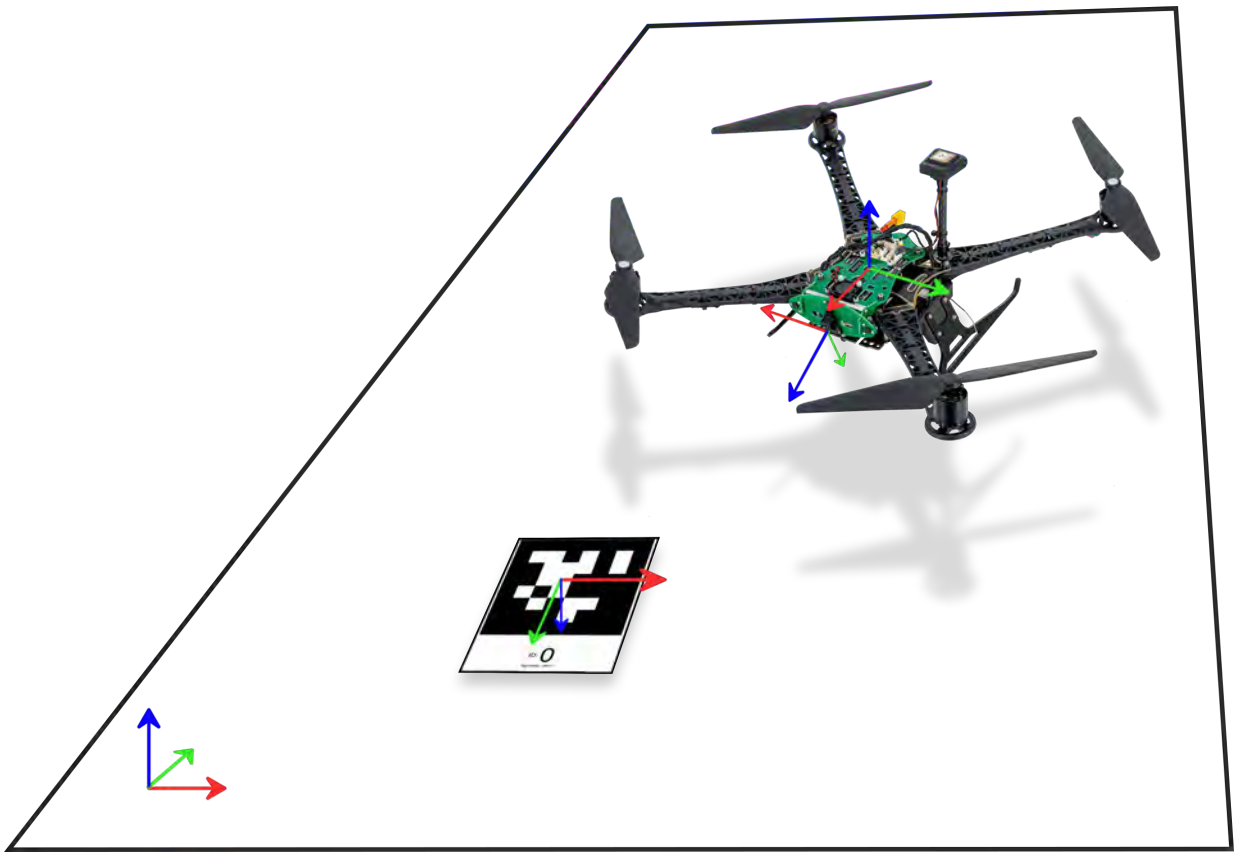


Figure 3.9: Visualization of different coordinate frames

the buckets affixed to the bucket configuration. The purpose of this project is to automate the task of the human pilot to drive their drone around the bucket configuration and peer into the buckets. This sort of behavior is meant to represent a first responder performing precise indoor maneuvers around debris in a disaster scenario, searching for signs of life among the rubble. This behavior is meant to develop an autonomous method for drones to perform first responders' tasks in emergencies like triage assessment, especially in complex scenarios where it's difficult to access the scene due to physical risks.

In triage assessment tasks in SAR, it assesses different vital signs such as breathing, bleeding and concsiousness [31], in which the evaluation of these signs are then used to prioritize injured people in different categories based on their conditions. However, for that to be done using a robotic system such as a drone, via a contactless device only using camera sensors, we need to drive the drone to operating distances based on the sensor platform, and the assessment algorithm requirements [32]. In other words, to get accurate results out of the inspection task, we need to make sure the drone can navigate to the known operating poses from the targets of interest autonomously. Hence, this section discusses the process of generating those poses of interest, recording them, then automating the process of executing those poses by the drone.

3.3.1 Data Preparation: Recording Poses of Interest

To effectively drive the drone to optimal inspection poses for viewing inside the buckets, our strategy involved defining a set of target positions. The process, adaptable to various applications, revolves around identifying precise drone locations where its tracking camera can clearly observe the bucket's interior. The method relies on establishing a consistent reference point,

similar to targeting a human's chest and face for pose estimation before navigating to an ideal observational distance. In our context, this fixed reference is the AprilTag's pose, centrally located on the bucket configuration. Thus, each desired pose is determined relative to the fixed AprilTag position.

To ascertain these critical poses, we employed a hands-on approach. The drone was manually positioned to ensure the tracking camera had an unobstructed view of each bucket's interior, as illustrated in Fig 3.5. Holding the drone steady in these positions ensured the AprilTag remained in the camera's frame, essential for accurate pose estimation.

The "generate_waypoints" program's primary function is to transform the detected AprilTag pose, received as a geometry_msgs/PoseStamped message, into a homogeneous transformation. This transformation represents the AprilTag's 3D pose in the drone's camera frame. The program then converts this pose from the camera frame to the drone's body frame, utilizing the formula:

$$^{B}T^{A} = {}^{B}T^{C}\,{}^{C}T^{A} \tag{3.2}$$

When the drone is correctly positioned, the user's prompt to save the pose initiates the recording of this data into a JSON file. This procedure results in a collection of 3D poses (homogeneous transformations) that describe each target position relative to the drone's body frame. These saved poses, defining the drone's spatial offsets from the reference point on the target, form the groundwork for the drone's autonomous navigation to the desired inspection points.

3.3.2 Autonomous Inspection Algorithm and Execution

The inspection routine initiates upon detection of a designated AprilTag. The drone's tracking camera, upon capturing this AprilTag linked to a specific bucket configuration, triggers a program that retrieves a JSON file corresponding to the AprilTag's TagID. This file, unique to each bucket configuration, lists sequential offset poses between the AprilTag and the drone's desired pose, expressed as homogeneous transformations. These transformations, termed ${}^{O}T^{A}$, represent the drone's position relative to the AprilTag at each specified offset.

To guide the drone to these positions autonomously, we convert these offsets into the local inertial frame. The desired offset in the inertial frame, ${}^{I}T^{O}$, is computed using a series of transformations:

$$ {}^{I}T^{O} = {}^{I}T^{B}\ {}^{B}T^{C}\ {}^{C}T^{A}\ ({}^{O}T^{A})^{-1} \tag{3.3} $$

This formula combines the drone's current pose in the inertial frame, the camera's pose relative to the drone, and the AprilTag's position relative to the camera, with the recorded offset. ${}^{I}T^{O}$ thus represents the drone's target position relative to the AprilTag, recalculated in the inertial frame.

The translational components of ${}^{I}T^{O}$ are relayed to the drone's autopilot via the mavros 'setpoint_raw/local' rostopic, which accepts mavros_msgs::PositionTarget messages. These messages specify yaw or yaw rate rather than a complete orientation. To determine the necessary yaw rate, a function, determine_yaw_rate, calculates the angular difference between the drone's current and desired yaw, adjusting the result within +/- [0,180] degrees and dictating the direction of yaw

adjustment. Implementing smaller yaw rates, as opposed to commanding direct yaw angles, was found to smooth drone movements, enhancing the stability and clarity of the inspection task. This smoothness is critical, particularly when transitioning from inspecting one bucket to the next, a process governed by a function that checks the drone's alignment with predefined tolerances for orientation and position.

The success of this inspection task heavily relies on the accuracy of the AprilTag's pose estimate. Any error in determining the location and orientation of this reference point could lead to incorrect drone positioning, undermining the inspection's effectiveness. In practice, the drone executed the inspection task effectively. However, an enhancement could include integrating feedback mechanisms, such as an object detection model to identify labels on the buckets. Due to the absence of pre-existing models for such specific targets, the subsequent section explores the development and deployment of a tailored neural network for detecting these markers on the drone's computer.

3.4 Integration of Object Detection into Autonomous Inspection

Object detection plays a pivotal role in SAR missions, where time is critical and accurate identification of targets is needed. To address this need, this thesis leverages You Only Look Once (YOLO) [33], a state-of-the-art object detection model known for its quick and precise performance. YOLOv5 stands out for its unified approach, processing images in a single run and thus providing the speed necessary for real-time applications on drones, like the M500 used in this project. This model which was originally trained on the diverse COCO dataset [34], has been retrained with a custom dataset of vision acuity markers. These markers, captured by the tracking

camera mounted on the M500 drone, serve as the labels affixed to the base of the bucket targets (refer to subsection 3.2.1 for a detailed discussion).

In the following subsections, we will delve into the details of the steps depicted in Fig. 3.10. Starting from the image data acquisition in subsection 3.4.1, progressing through the model training, validation, and the deployment for on-board testing of the model in subsection 3.4.2.

3.4.1 Vision Acuity Marker Detection and Data Acquisition

Data collection was conducted using the M500 drone, specefically with the tracking camera (M0014 module[6]). The process involved capturing a broad range of images under different lighting conditions to account for different operational scenarios. The drone was used both in flight and held by hand to record rosbags of the tracking camera's image stream, focusing on the markers at the bucket bottoms. The goal was to collect a comprehensive dataset which is important for developing an effective object detection system. The raw images were then extracted from the recorded rosbags using Python, providing a dataset with a total of 1511 images.

The precision in data labeling is vital in object detection models, especially for single-shot models such as YOLOv5 that rely on precise bounding box coordinates for accurate object identification. Using Roboflow, each image was manually annotated with bounding boxes drawn around the markers and labeled for eight distinct classes. The annotations need to be precise to ensure accurate results from the model to detect and classify various objects. As a preprocessing step, all the images were then resized to a uniform 416x416 pixels to meet YOLOv5 model input requirement. Fig.3.11 shows a visual example of this process.

Moving to Data augmentation, which is a critical step in the development of robust object

[6]Tracking Sensor Datasheet

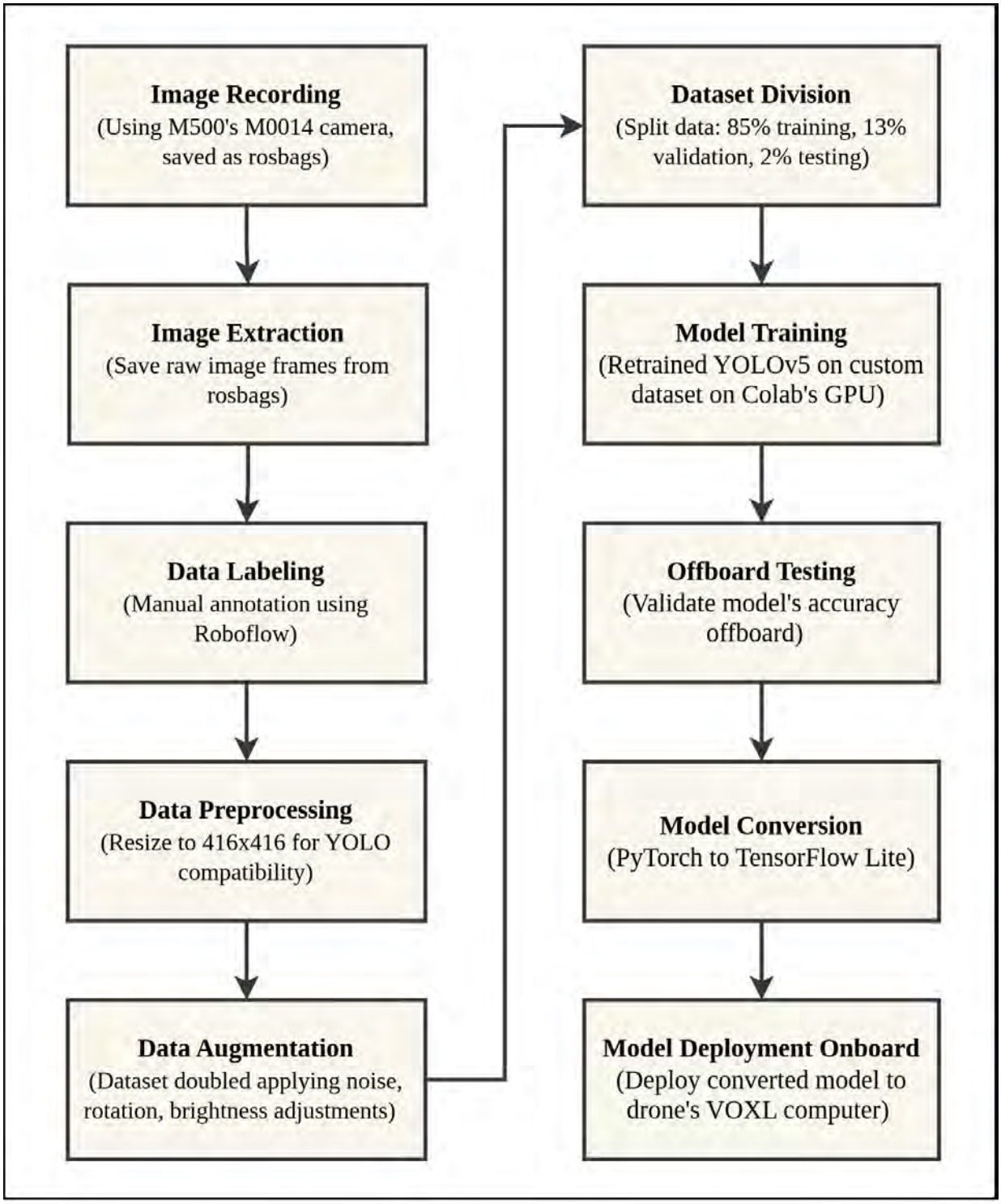


Figure 3.10: Workflow for object detection model training and deployment on the drone

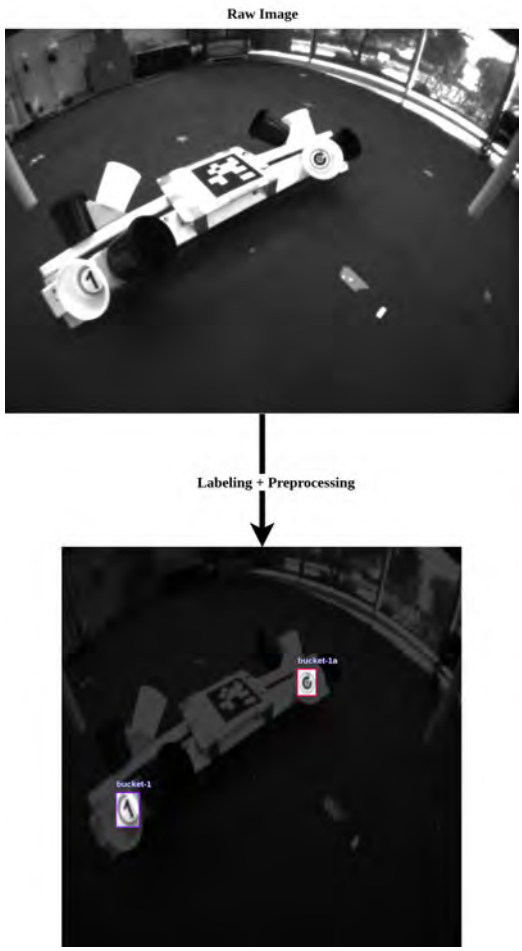


Figure 3.11: Example of data labeling and preprocessing for YOLOv5

detection models. As shown in Fig. 3.12, by applying transformations such as noise addition, rotational adjustments, and brightness-contrast variations, the dataset's diversity is significantly enhanced. The specific augmentation parameters used on this dataset included:

- Noise injection up to 1.96% of pixel variation
- Crop zoom ranging from 0% to 28%
- Random rotation between -20° to +20°
- Brightness alteration from -17% to +17%

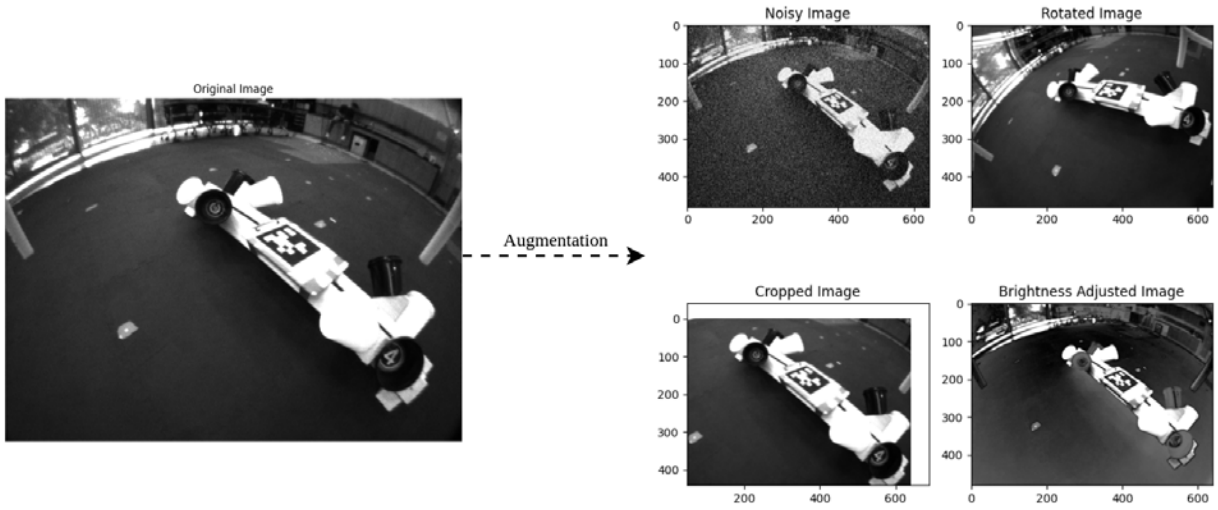


Figure 3.12: Data augmentation example on one image

As shown in Fig. 3.13, post-augmentation, the dataset expanded from 1511 images to 2626 images. It was partitioned into subsets for training (85%, or 2230 images), validation (13%, or 354 images), and testing (2%, or 42 images). This division was planned to present the model with a variety of scenarios during training, while the validation set is for fine-tuning the model's performance while in training, and testing set is for assessing its final accuracy on unseen data.

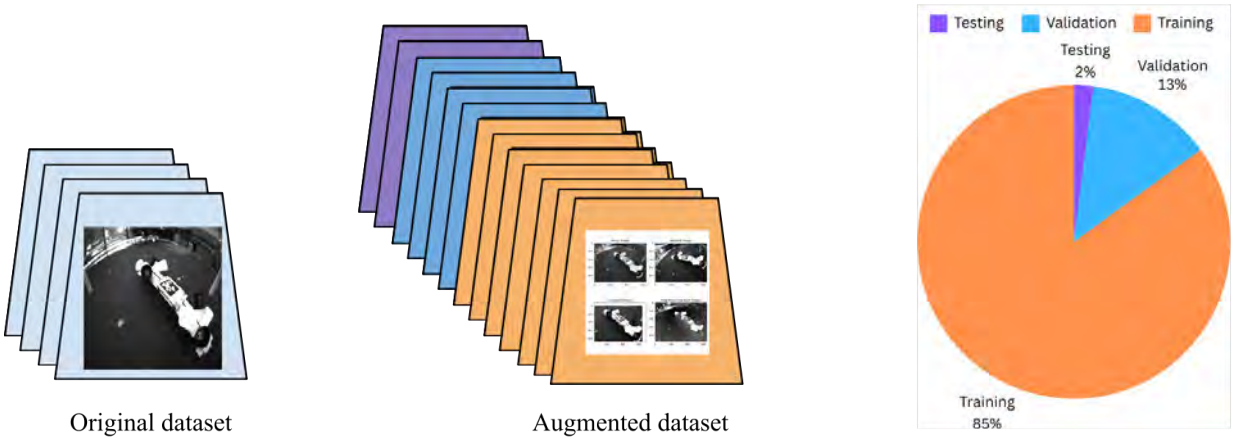


Figure 3.13: 2x Augmentation of original dataset from 1511 to 2626. Division into Training, Validation and Testing

3.4.2 Neural Network Training, Validation, and Onboard Deployment

3.4.2.1 Model Training Details and Analysis

For this project, training the neural network model was executed on Google Colab, leveraging its GPU resources, and specifically the training was done on Tesla T4 GPU, employing PyTorch version 2.2.1. The dataset, sourced and preprocessed as described earlier in subsection 3.4.1, was then imported into Colab environment using the Roboflow API.

With a focus on achieving a balance between speed and accuracy, the YOLOv5s [35] architecture was chosen as the backbone for the neural network. YOLOv5s is the small variant of YOLOv5. It was an ideal match for the computational constraints of the onboard drone hardware due to its compact size and rapid inference ability. Training started with pretrained weights, and the process of transfer learning started to adapt the model to the features of the vision acuity markers. Training was conducted over 150 epochs, batched in groups of 16, with each image resized to a 416x416 resolution to comply with the model's input specifications. Fig. 3.14 presents a batch of 16 training images, which have been subjected to various augmentation methods as shown. Also, Fig. 3.15 showcases a sample batch from the validation dataset, showing the predicted label during training along with the prediction confidence. Note that the validation set is not augmented since the role of it is to mimic the potential real-world scenario as closely as possible.

The assessment of the model's performance is depicted in several key plots as shown in Fig. 3.16, which reveals significant insights into the model's training and validation phases. The first row of plots represents losses and metrics across the training epochs. Specifically, the first

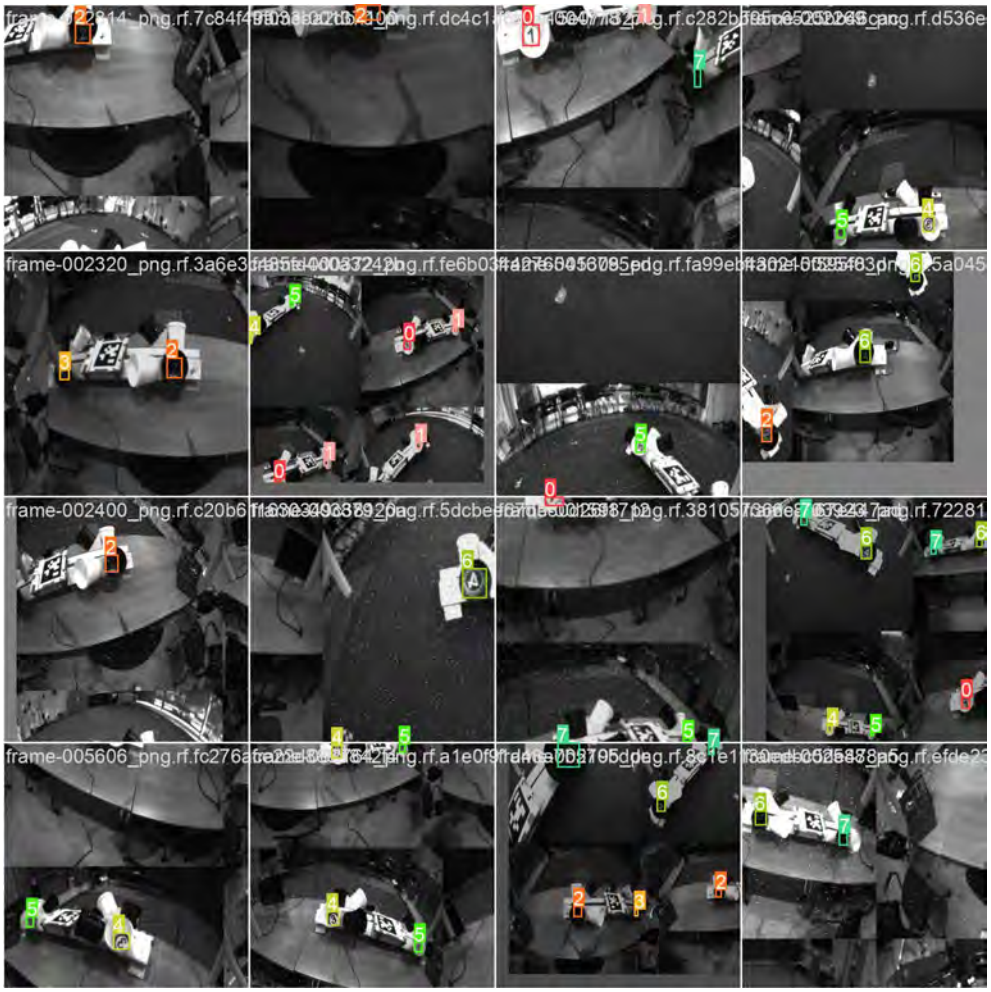


Figure 3.14: Sample training batch with augmented images

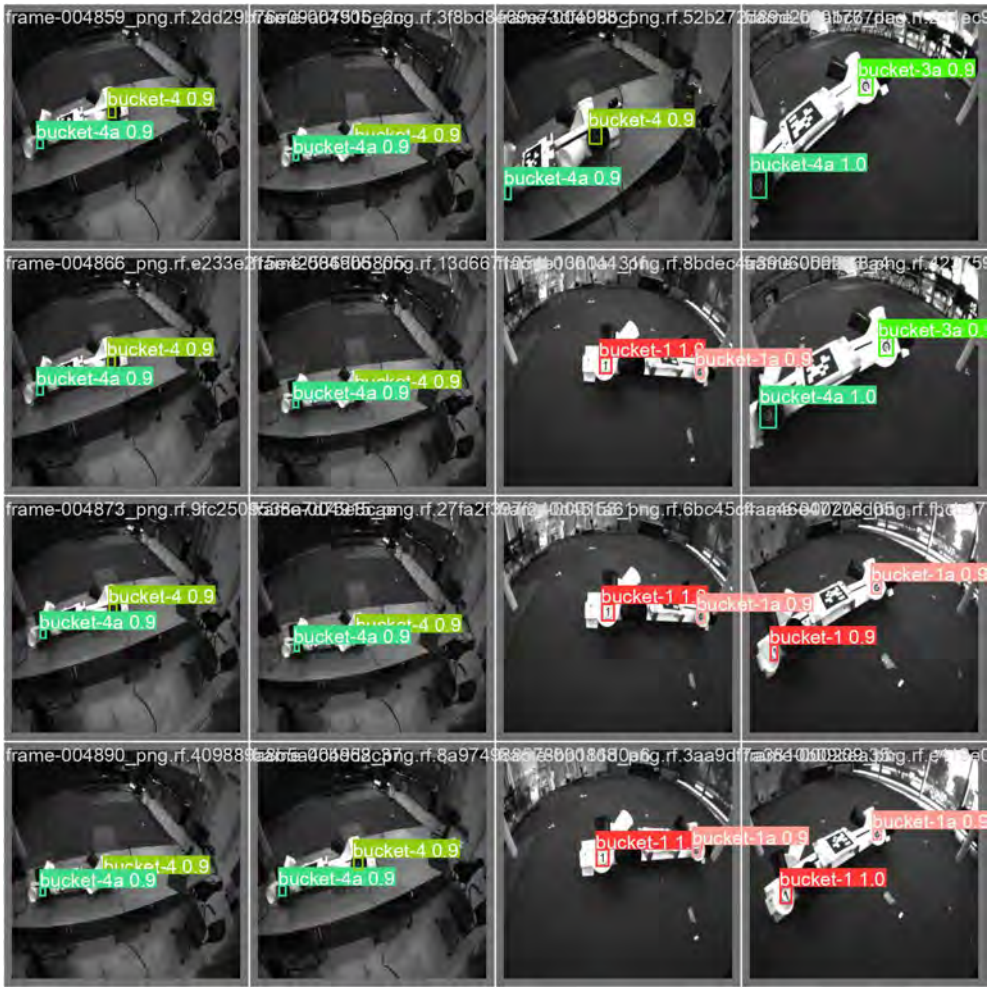


Figure 3.15: Validation batch sample with prediction confidences

plot captures the training box loss, which depicts how well the model predicts the bounding box coordinates for each object. As seen, the trend is descending which suggests an increase in the model's proficiency at locating objects within the image frame as training epochs progress. The training object loss plot reflects the model's improvement in distinguishing objects from the background, again with decreasing trend indicating the model learning to reduce false positives over time. The third plot; the class loss plot, points to the model's growing accuracy in classifying objects into their corresponding classes correctly. The fourth and fifth plots on the top row track precision and recall over training epochs, respectively. Precision measures the model's ability to

return only relevant instances, which shows a stabilizing trend at very high values, suggesting few false positives. Recall indicates the model's success in finding all relevant instances, which also stabilizes at high values to imply the model is becoming increasingly capable of detecting all objects of interest. The second row of plots in Fig. 3.16 shows similar narrative but for the validation data, where the first three plots correspond to validation box loss, validation object loss, and validation class loss. These metrics demonstrate the model's ability to generalize from its training data to new, unseen images from the validation set. Last two plots provide a quantification of model accuracy through mean Average Precision (mAP) at two Intersection over Union (IoU) thresholds: 0.5 and 0.5:0.95. The mAP at 0.5 is a measure of accuracy at a single threshold, while the mAP across 0.5:0.95 gives a more comprehensive assessment over a range of object sizes and overlapping scenarios. Appendix A provides further details on the precision and recall analysis of the model's performance.

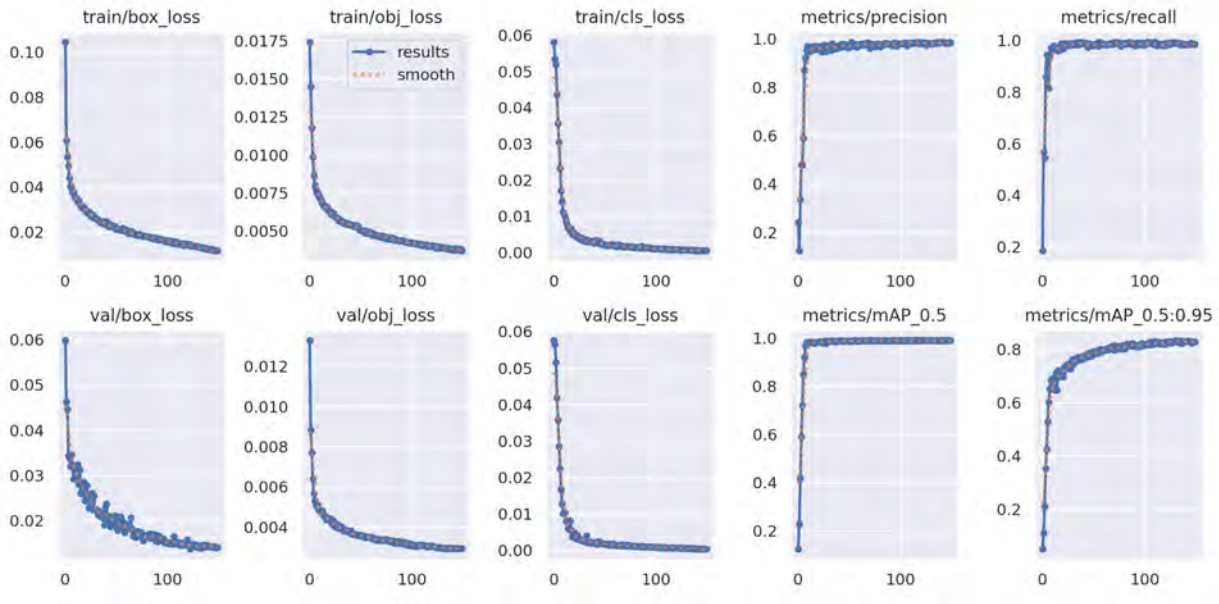


Figure 3.16: Training and validation losses illustrating the model's learning efficiency across epochs

3.4.2.2 Testing the Model Offboard

Upon completion of the training, the model's best-performing weights were exported for an offboard testing phase. This phase employed YOLOv5_ROS[7], which adapts the input of `detect.py` from `ultralytics/yolov5` to process `sensor_msgs/Image` from a ROS node. This setup allowed for real-time analysis by subscribing to the tracking camera's feed, published by the ROS node running on the drone, and processing the imagery on a local machine. It outputs resized images with the prediction labels, and the corresponding detections as a custom message that encapsulates the information of the bounding boxes details, including the `class_name`, the confidence score, and the box coordinates within the image plane.

This process introduced a measure of latency due to data transmission across the network; however, it was a vital step in confirming the model's real-time detection capabilities. The system maintained an average inference time of 175.47 ms, effectively demonstrating the model's aptitude for real-time application. This step was critical in verifying the transferability of the training to practical application scenarios prior to deploying the model to the onboard computer for faster inference.

3.4.2.3 Model Onboard Deployment on VOXL

To transition from offboard inference to onboard utilising VOXL's GPU on the drone, the first step was converting the best-performing model from PyTorch (.pt) to TensorFlow Lite (.tflite) format to be compatible with `voxl-tflite-server`[8] running onboard. This server operates as a systemd background service as hardware-accelerated TensorFlow Lite environment. As

[7] YOLOv5 ROS wrapper
[8] VOXL tflite server documentation from ModalAI.

of the time of writing this, VOXL TFLite server supports two TensorFlow Lite build versions (delegates): `apq8096` and `qrb5165`. This project uses the `apq8096-tflite` version with the GPU delegate enabled on the M500 drone's VOXL1 computer. The conversion from `.pt` to `.tflite` necessitates model compression, quantization, and modification of the tflite-server source code to accommodate the custom neural network's input and output tensors.

Post-training quantization was part of this process, which optimized the model for the drone's GPU. It reduced model size and computational complexity to make it usable for low-latency, real-time operations on the drone. We can also utilize the existing `export.py` script provided by YOLOv5's team to export the `.pt` model to a `.tflite` model. However, compatibility with the TensorFlow Lite version running on VOXL is not always guaranteed. If the exported `tflite` model does not work with the tflite-server running on VOXL, then retraining the model using the TensorFlow version matching the VOXL SDK version should be considered.

The next crucial step involved customizing the `voxl-tflite-server`'s postprocessing functions, to accommodate the model's output tensor structure and dimensions, which has been known by visualizing the converted model's layers using tools like Netron[9]. These changes made sure the model's output, such as class IDs, bounding box coordinates, and confidence scores were accurately processed and understood by the drone's system.

Additionally, the `InferenceHelper` class was adjusted for this specific model, where it was tailored to parse the model's detections. This included providing inference outputs such as drawing bounding boxes on the output images, annotating them with labels, and compiling detection metadata for subsequent use. Each detection was recorded with a timestamp for temporal tracking. After making these changes to the original TFlite server running on VOXL, it

[9]Netron: tool to visualize neural networks

was then built using a dockerized build environment that mimics the build environment used to build the original packages and services on VOXL. After building the custom package and with the adjustments made, the model was successfully deployed onboard the VOXL. This has been verified from the metadata output, which correctly showed all necessary details for each detection. These metadata could be subscribed to over a ROS framework for real-time decision-making during flight operations, which enables the drone to identify and respond to visual cues autonomously.

Comparing offboard and onboard performance, there was a notable improvement in the model's onboard inference time, dropping to an average of 71.3 ms, compared to 175.47 ms offboard. This reduction highlighted the success of the onboard deployment, indicating the model's readiness for real-time operational use and the compatibility within the drone's hardware limits. Fig.3.17 shows a snapshot of the drone flying to inspect the buckets, while running onboard custom detection model and at the same time running offboard inferences on my local machine, analyzing the published camera stream from the drone. The figure shows the lag in the offboard detection compared to the onboard inference which is coping with the drone's actual position at that time instance.

3.4.2.4 Limitations and Future Improvements

Despite achieving positive outcomes in model deployment and performance, we encountered several challenges and limitations. A primary issue was the drone's tracking camera's limitations, specifically its low resolution and monochromatic output. Tracking sensor is primarily used for drone's localization by utilizing its fisheye lens and the downward orientation to extract

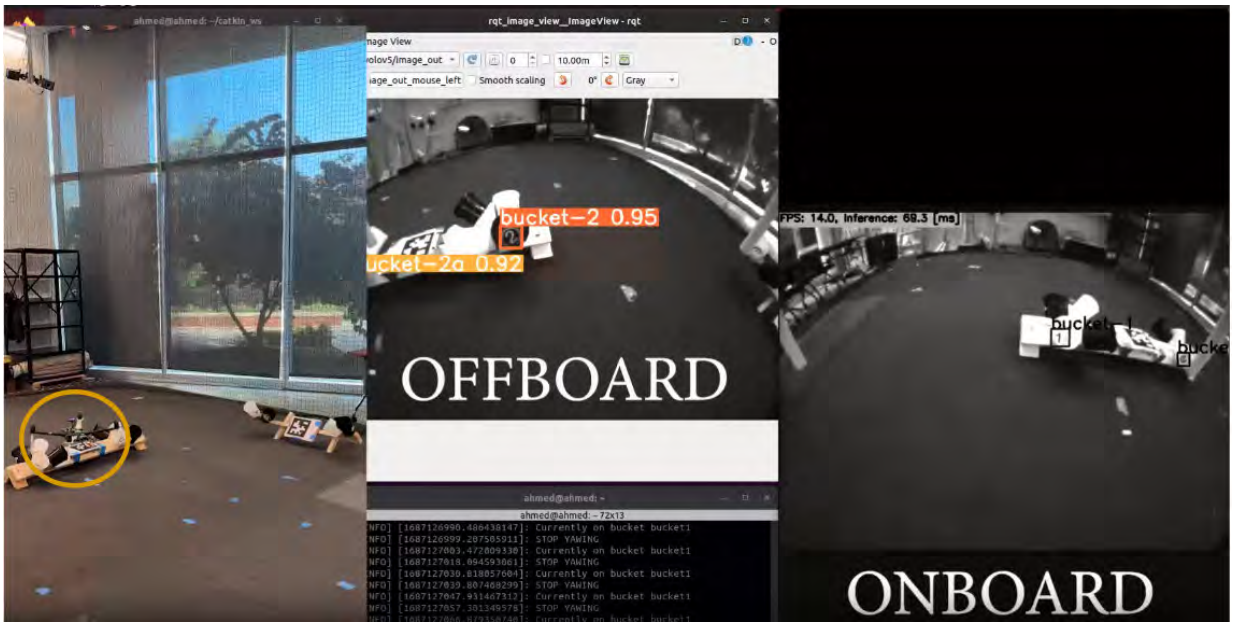


Figure 3.17: Difference between onboard detection streamed to VOXL portal (right), and offboard detection running on my local machine (middle). Drone's actual location is shown on the left

more features from the environment for better localization. However, this project utilized it as the primary image source leading to a lack of color detail in grayscale images. This affected the model's ability to differentiate between visual cues, especially the darker labels in low light conditions. For future enhancements, integrating a higher-resolution color camera is recommended, which would improve the model's efficiency in processing and interpreting visual data. It's important to note that the steps for implementing the neural network model on the drone's onboard system would remain exactly as outlined in this section. Additionally, an upgraded camera system would require a more powerful companion computer to manage the data output from the advanced sensors effectively.

Another aspect requiring attention is the camera's fisheye lens. In future applications involving precise localization based on object detection, image rectification will be essential for aligning the camera's image accurately with the drone's spatial surrounding. This adjustment will

be crucial, especially in missions requiring exact matching of visual cues with real-world spatial information to re-orient the drone for instance. Therefore, calibrating the camera's intrinsic parameters will be an integral part of the framework to rectify the images used for model training before starting the training. This approach ensures that the model operates optimally on rectified image streams, which will enhance the accuracy the detections, and will help the system to extract spatial data from the image plane, probably by incorporating a depth sensor.

In the next section, the integration of the various modules discussed in this chapter will be elaborated upon, demonstrating how they collectively run together to complete the autonomous mission on the drone's hardware.

3.5 Search and Inspection Integration

After discussing how each module works separately in the previous sections, the subsystems integration to achieve the autonomous mission is discussed in this section. Fig. 3.18 shows a high level system architecture of the proposed system. This architecture is the one commonly used for autonomous systems, where it breaks down the overall system into perception, planning and control. The first layer is the perception system, which takes the the raw image stream coming from the tracking camera and the IMU sensor readings as inputs.

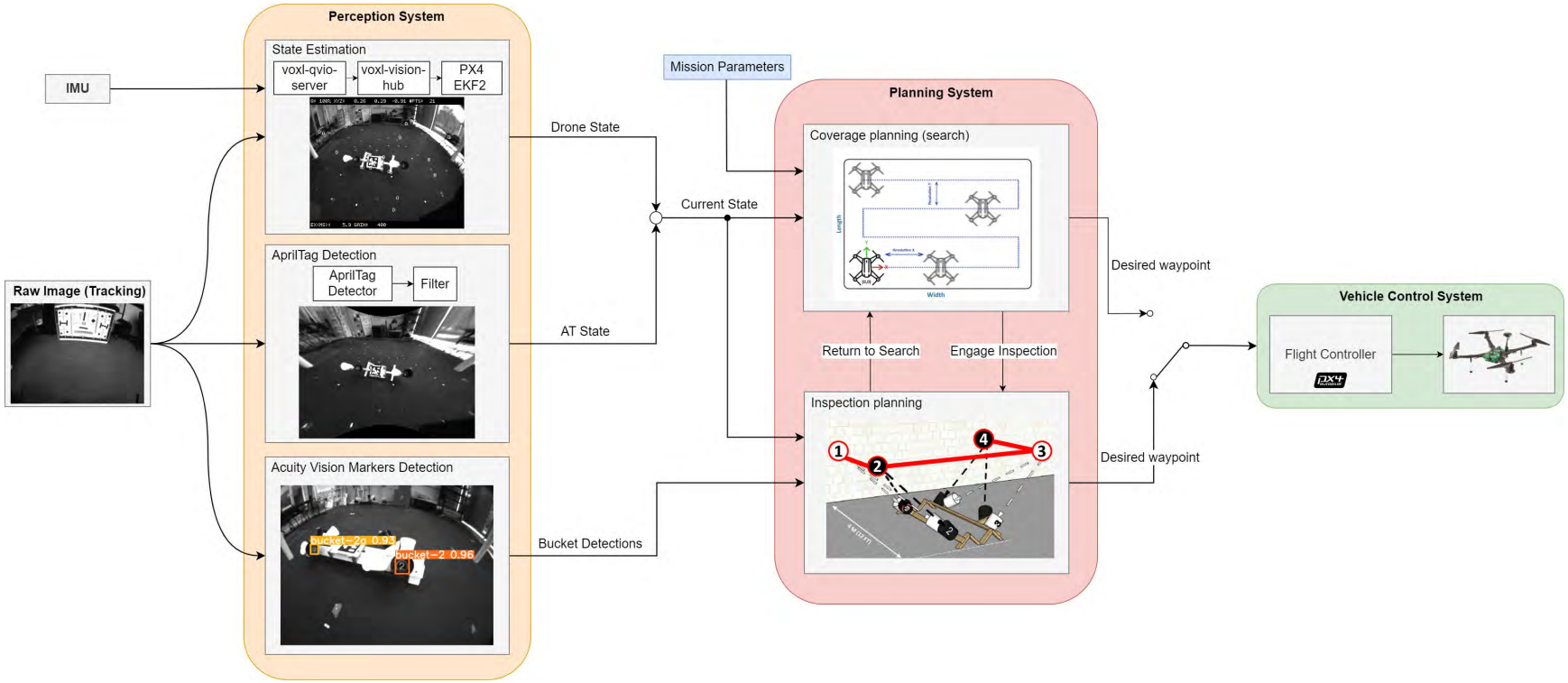


Figure 3.18: System integration of different modules

The state estimation subsystem fuses the gyroscope and accelerometer readings from the IMU, along with the features tracked from the camera feed, to estimate the relative position and orientation of the drone in the local inertial frame using Visual Inertial Odometery (VIO). On VOXL, the ‘voxl-qvio-server‘ is the pipeline responsible for calculating the VIO data, where qvio corresponds to Qualcomm VIO, which refers to the technology developed at Qualcomm to work efficiently on resource constrained devices such as smartphone. This made it convenient for use on the drone's companion computer which has the snapdragon processor. This is suitable for the Qualcomm algorithm since it uses Qualcomm Snapdragon processor's optimization for rapid computation time with low memory usage, by utilizing vector processing and parallel computation abilities. The combination of the landmarks extracted from the camera with the inertial sensor measurements using EKF framework, it is able to estimate the device pose along with the calibration and estimation statistical parameters such as biases, variances and scale factors. The result of QVIO is then passed to voxl-vision-hub, which sends this data to PX4's EKF2 framework. The final pose estimate that we use for the drone's pose estimate of itself is the final output from PX4's EKF2, which we interface with using MAVROS. This output, referenced in Fig. 3.18 as the drone state, is the representation of the drone's 3D pose in the free space, which consists of:

$$[X, Y, Z, q_x, q_y, q_z, q_w]$$

Position data are in the local NED/FRD for outdoor/indoor respectively. The quaternion data represents the body orientation in the free space as the rotation from the FRD body frame to the local NED inertial frame. This also aligns with the IMU frame on VOXL and with the QVIO

data on VOXL. However, working with MAVROS, it interfaces this to ENU for the local inertial frame and FLU for the body frame.

The seconds sub-module is the AprilTag detection subsystem. As explained, it filters the readings from the detections of the AprilTag along with doing homogeneous transformation to eventually getting a final pose of the AprilTag in the local inertial frame, in addition to the detected AprilTag's ID. Hence, the output of this module is referenced as AT state, which consists of:

$$[X, Y, Z, q_x, q_y, q_z, q_w, ID]_{AT}$$

Then the two outputs; drone state and AT state, are combined together to form one state vector referenced as the 'current state' , which keeps track of both the drone state and the AprilTag state. This combined state is then used as input to the planning layer.

The tracking camera also feeds the third submodule in the perception layer which is the acuity vision marker detection model. As discussed in Section 3.4, the output of the custom trained object detection model is the bounding box metadata for the detected objects in the image frame. The information that we use in this framework is the detected class name along with the detection confidence level. These information are then used by the inspection routine subsystem in the planning layer.

The planning layer consists of the interplay between the search (coverage) planning and the inspection routine. Both of the submodules take the combined state as one of the inputs, and both of them outputs a desired setpoint for the flight controller to execute. Detailed in Section 3.1, the search algorithm takes the desired mission parameters as an input along with the combined state,

to plan the coverage path of the area of interest, which consists of the sequence of waypoints that cover the entire area. The mission parameters are the desired search altitude, dimensions of the area of interest, and the resolution along the x and y axis. On the other hand, the inspection routine takes, beside the combined state, the output detections from the acuity vision marker model to alter the inspection routine and serves as some sort of feedback while following the inspection poses loaded from the JSON file. The logic that governs the interplay between the search and inspection in the autonomous system is depicted in the swimlane flowchart in Fig. 3.19

To illustrate, the drone starts the coverage by following the coverage path, while looking for new AprilTag to detect. Once a new AprilTag is detected, the program switches from search mode to inspection mode, and the drone stops to filter the pose of the detected AprilTag for better localization of the target. Then, it loads the JSON file corresponding to the detected AprilTag ID, which contains the pre-recorded poses of the drone's body with respect to the AprilTag for that particular target. The drone then goes to a closer proximity of the first bucket as commanded, and once it becomes within an acceptance tolerance from the desired pose, it starts counting both the number of detections of the current bucket, and time spent at current bucket. The drone stays at current bucket until either:

- It reaches or exceeds a user-defined value for minimum number of detections per bucket AND if the running average of the detection confidence for the current bucket is more than or equal a defined confidence threshold.
- It spent more than or equal the user-defined maximum time allowed for it to stay at one bucket, regardless of the number of detections or their average confidence.

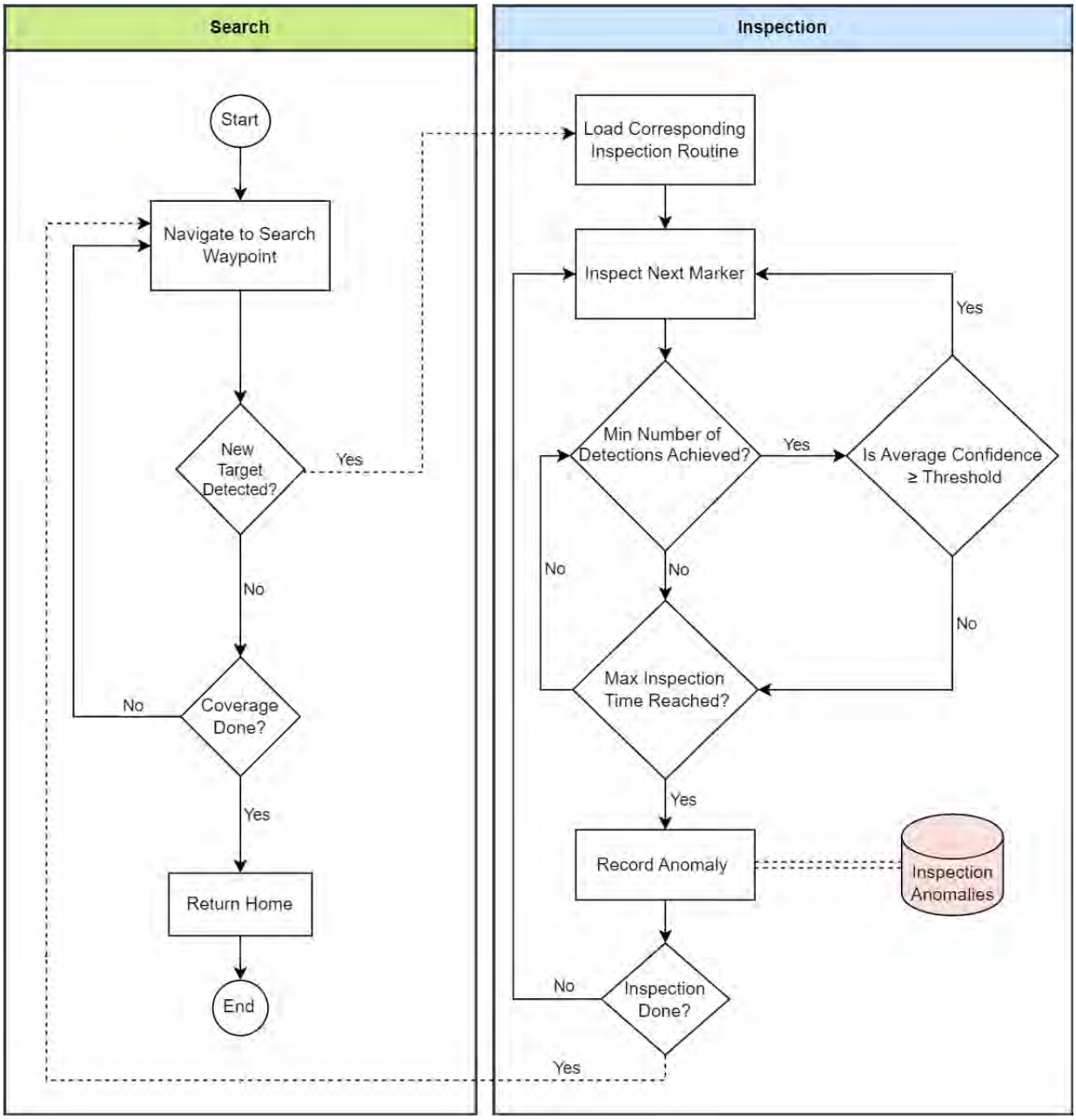


Figure 3.19: Flowchart for the framework between search and inspection

If the drone reaches the allowed average confidence level from a pool of detections which is more than or equal the allowed number of detections, this means that the drone is quite sure about the current inspected bucket, and there is no need to stay for more time at that bucket, hence it moves to the next one. If it does not reach that allowed confidence or it does not even get the minimum number of detections, it will move on to the next bucket if time at current bucket is more than or equal the maximum allowed inspection time per bucket. However, in that case, the program saves this point as an anomaly that needs further investigation and review. The inspection anomalies are the point where the drone struggled to reach to, and at which the model struggles and was not very confident about that detection. It saves that anomaly as the drone's pose at that point along with other relevant information about the detection such as the number of detections and confidence. The drone autonomously covers all planned inspection poses as outlined in the JSON file it loaded, then once the poses are finished, it flags that bucket configuration with its corresponding AprilTag as inspected, so that it does not inspect it again if it detected that apriltag. Also, it turns the inspection routine off and switches back to the search routine, where the drone goes on to continue following search path from where it left off for inspection. The loop continues until the entire area is covered, and all targets are inspected, then the drone goes back to its takeoff location.

Chapter 4: Autonomous Navigation in Cluttered Environment

This chapter shows the development and integration of path planning algorithms designed for UAV autonomy in cluttered indoor environments. It outlines the implementation details, the adaptation of one of these algorithms to the specific challenges posed by indoor settings, and its integration within a software-in-the-loop simulation framework. It also outlines the potential of adapting it to real-world tests.

4.1 Review of Existing Path Planning Algorithms

4.1.1 Comparative Analysis of Relevant Packages

The selection of the path planning package is important, since it impacts the safety of the drone during the mission. A comparative analysis of state-of-the-art path planning solutions reveals different use cases where each algorithm brings unique strengths to the table.

4.1.1.1 FastPlanner

FastPlanner is designed for rapid and efficient trajectory generation, optimized for quadrotors in high-speed flight scenarios [36]. It employs a kino-dynamic path searching algorithm within a voxelized map, which seamlessly transitions to a trajectory optimization process. The

planner's remarkable feature lies in its dual-phase optimization strategy, which initially handles geometric path planning and subsequently refines the path considering dynamic constraints. It stands out for its ability to quickly adapt trajectories, ensuring minimum jerk – crucial for reducing wear on drone components and enhancing flight stability.

4.1.1.2 Faster

The Faster trajectory planner introduces an innovative Mixed Integer Quadratic Program (MIQP) formulation, enabling the planner to operate within both known and unknown spaces [37]. Its distinct capability to generate a safe backup trajectory at each replanning step ensures robustness against unforeseen environmental changes. Faster's architecture is conducive to agile flight in unknown cluttered environments, showing promise in environments where the known space (F) is minimal compared to the unknown (U).

4.1.1.3 EGO-Planner

Building upon the strengths of FastPlanner, EGO-Planner eliminates the need for a pre-built Euclidean Signed Distance Field (ESDF) [38]. It instead employs a collision-free guiding path to formulate the collision cost. This method is lightweight and bypasses redundant computations, thus accelerating the planning process. EGO-Planner is adept at real-time adjustments, dynamically reallocating trajectory times to ensure dynamical feasibility, an essential feature for a drone's quick responsiveness in complex environments.

4.1.1.4 EGO-Swarm

EGO-Swarm is the multi-agent evolution of the EGO-Planner, specialized for scenarios involving multiple drones and dynamic obstacles [39]. It integrates the collision avoidance capability of EGO-Planner with inter-agent coordination mechanisms. The planner's utility is exemplified in its swarm application, where each drone autonomously and collectively contributes to environmental exploration and task execution.

4.1.1.5 Mader

Lastly, Mader represents a trajectory planner tailored for multi-agent systems immersed in dynamic environments [40]. It is noted for its trajectory planner's flexibility in a shared space, ensuring each agent's path is computed considering the others' planned trajectories, thus minimizing collision risks and optimizing flow within the operational area. The planner is resilient, able to recalibrate its trajectory planning in response to environmental changes, making it highly applicable to real-time operations where environmental predictability is low.

Each of these packages demonstrates significant advancements in the field of autonomous drone navigation. However, for the purpose of this project, the chosen planner must exhibit not only real-time responsiveness and robustness in cluttered environments but also compatibility with the drone's onboard computing constraints and the operational goals. The analysis suggests that a planner offering a blend of speed, dynamical feasibility, and adaptability to both solo and swarm operations, along with an efficient computational footprint, would be most aligned with the project's requirements.

4.1.2 Chosen Planner Rationale

After a review of the field, FastPlanner was identified as the ideal fit for the mission's needs. Its selection was based on its computational efficiency and its proficiency in generating dynamically smooth trajectories. The smoothness of the trajectory is significant in complex environments as it directly impacts the safety and reliability of the drone's navigation. FastPlanner-'s trajectories, optimized through a two-stage process including kinodynamic pathfinding and B-spline refinement, offer a balance between speed and maneuverability without compromising on safety. Another significant factor contributing to the choice of FastPlanner is its extension, FUEL [41]. This development enhances the planner's functionality, enabling efficient exploration and autonomy in unknown spaces, which is a core requirement for any mission in SAR.

While EGO-Planner provides quick computation by foregoing the construction of an ESDF, it yields less smooth trajectories. Such trajectories can increase the risk of abrupt movements and potential collisions.

4.2 Detailed Description of the Planning Algorithm

This section outlines the technical framework of FastPlanner for trajectory generation, combining kinodynamic path searching, B-spline optimization, and iterative time adjustment. This framework does not impose strict velocity and acceleration constraints, which allows for faster area coverage. It begins with kinodynamic pathfinding to quickly identify optimal paths, followed by B-spline optimization for safety, and ends with time adjustments to ensure trajectories meet practical kinodynamic limits.

4.2.1 Hybrid A* Algorithm in Kinodynamic Path Finding for Quadrotors

The algorithm begins with the initial state of the quadrotor and an open list of nodes to be explored. Each node represents a possible state of the quadrotor, defined by its position and velocity. The algorithm expands nodes by applying a set of motion primitives, which are short segments of trajectory calculated based on the current state and a set of discretized control inputs.

As the algorithm explores each new state, it calculates g_c, the cost so far, by summing the edge costs of the path taken to reach this state. The heuristic cost h_c is then computed to provide an estimate of the cost to reach the goal from this new state. The total cost f_c (sum of g_c and h_c) is then used to evaluate the optimality of the new state.

If a new state falls into a grid cell that already contains another state with a higher cost, the algorithm replaces the existing state with the new, more optimal one. This pruning (deleting) ensures that only the most efficient paths are considered. The search continues, expanding states and evaluating costs, until it reaches the goal or exhausts all possible paths [36].

4.2.1.1 The Hybrid A* Algorithm

The hybrid A* algorithm differs from the conventional A* search. While the conventional A* algorithm excels in finding optimal paths in a discrete space, it falls short for non-holonomic, underactuated systems like quadcopters operating in 3D environments. In such systems, strictly adhering to discrete, straight-line paths from the A* search can lead to infeasible trajectories due to the sudden directional changes which do not align with the physical capabilities of the quadrotor.

Hybrid A* addresses this by integrating the vehicle's kinematic constraints directly into

the path planning process [42]. This integration allows for smoother transitions between path vertices, accommodating the non-holonomic nature of quadrotors. Unlike conventional A*, where transitions between nodes are straightforward but potentially infeasible, hybrid A* ensures that each move is dynamically executable by the quadrotor.

4.2.1.2 Motion Primitives

At the core of the hybrid A* algorithm for quadrotors are motion primitives, which are small predefined maneuvers based on the quadrotor's dynamic model. The concept of differential flatness is integrated here, which allows the representation of a quadrotor's states and inputs using a minimal set of "flat outputs" and their derivatives. Typically, for a quadrotor, these outputs include the 3D position of the center of mass (x, y, z) and the yaw angle (heading). This simplifies the process of generating smooth trajectories for the drone to follow. It is worth noting that yaw is not included in the trajectory representation for the motion primitive segments in [1], however, the smoothness is further improved by spline optimization later. The segments are represented in the 3D space as a series of polynomial functions of time [36], where

$$p(t) := [px(t), py(t), pz(t)], \quad p_\mu(t) = \sum_{k=0}^{K} a_k t^k \tag{4.1}$$

with $\mu \in \{x, y, z\}$, representing the x, y, and z coordinates of the quadrotor's position over time, motion primitives are generated by varying these polynomial functions using discretized control inputs U_D (acceleration in each direction) over a fixed short time interval τ, to expand the search nodes in a manner consistent with the quadrotor's kinematics.

4.2.1.3 Cost Function

In path planning, cost functions play the role of guiding the search towards the goal. Hybrid A* utilizes a cost function that balances the actual cost of the path (cost so far) and the estimated cost to reach the goal (heuristic). The actual cost, g_c represents the sum of the costs incurred along the path from the start state to the current state. The heuristic cost, h_c is an estimate of the cost from the current state to the goal, guiding the search efficiently toward the objective by approximate the remaining cost, but without overestimating it.

The cost of a motion primitive, or the edge cost, is computed as [36]:

$$\mathcal{J}(T) = \int_0^T \|u(t)\|^2 dt + \rho T \tag{4.2}$$

where T is the duration, $u(t)$ is the control input, and ρ is a weighting factor. This function reflects both the control effort and time, which ensures efficiency and responsiveness. Hence, g_c is calculated for J number of primitives expanded from previous step as [36]: $g_c = \sum_{j=1}^{J}(u_{d_j}^2 + \rho)\tau$.

The heuristic cost h_c is calculated using Pontryagin's Minimum Principle, a method from optimal control theory used to find the minimum control effort to reach a state. The principle allows the derivation of a closed-form solution for the minimum cost trajectory from the current state to the goal state. For each state component (say, x, y, z), the problem is solved, and the resulting cost is summed up to get the heuristic cost h_c. Mathematically, it involves solving a boundary value problem where the boundary conditions are the current state and the goal state. Finally, f_c is defined as the sum of g_c and h_c. It represents the total estimated cost of a path through the current node to the goal, which is a fundamental metric for the A* algorithm to

prioritize which nodes to explore next.

4.2.1.4 Analytic Expansion

To expedite the search process, the algorithm employs analytic expansion. This technique attempts to directly connect the current node to the goal with a collision-free and dynamically feasible path. If successful, it bypasses the need for further expansion, significantly speeding up the search. This is particularly effective in sparser environments, where direct paths to the goal are more likely to happen.

4.2.2 B-Spline Trajectory Optimization

After finding a feasible path via the kinodynamic search, the trajectory is refined for smoothness and safety by leveraging B-spline curves, which not only smooth the path but also ensure it steers clear of potential obstacles.

4.2.2.1 The Use of Uniform B-Splines

Trajectory refinement employs uniform B-splines, or Basis splines, which are mathematical representations used extensively in curve and surface modeling [reference about B-splines]. B-splines are characterized by a series of control points $\mathbf{Q}_0, \mathbf{Q}_1, \ldots, \mathbf{Q}_N$ and a knot vector $t_0, t_1, ..., t_M$. The control points determine the shape of the B-spline, while the knot vector, which consists of a non-decreasing sequence of parameter values, influences how the spline interpolates the control points. In uniform B-splines, the spacing between the knots in the knot vector is constant.

The curve at any time t is computed by [36]:

$$\mathbf{p}(s(t)) = \mathbf{s}^{\top}(t)\mathbf{M}_{p_b+1}\mathbf{q}_m, \tag{4.3}$$

where $\mathbf{s}(t) = [1, s(t), s^2(t), \ldots, s^{p_b}(t)]^{\top}$ represents the spline's basis functions. The matrix M_{p_b+1} is linked to the B-spline's degree p_b, and it controls how the spline behaves and responds to the positioning of the control points.

4.2.2.2 Kinematic Constraints via Convex Hull Property

The convex hull property of B-splines guarantees that the spline is contained within the convex shape formed by its control points. This geometric feature is vital for incorporating kinematic constraints like velocity and acceleration into the spline. The velocity $\mathbf{V}_i$ and acceleration $\mathbf{A}_i$ at a control point are derived from the positions of the spline's control points as [36]:

$$\mathbf{V}_i = \frac{1}{\Delta t}(\mathbf{Q}_{i+1} - \mathbf{Q}_i), \mathbf{A}_i = \frac{1}{\Delta t}(\mathbf{V}_{i+1} - \mathbf{V}_i), \tag{4.4}$$

where Δt is the time difference between consecutive knots. Adjusting these control points directly impacts the B-spline's velocity and acceleration to enable precise control over the quadrotor's movement.

4.2.2.3 Collision Avoidance through Convex Hull Analysis

Ensuring a collision-free trajectory involves analyzing the B-spline's convex hulls. The proximity of these convex hulls to potential obstacles is assessed to ensure safety. If a part of

the convex hull is too close to an obstacle, the corresponding control points are modified. These adjustments may involve repositioning the control points or changing the parameters of the spline to expand or contract the convex hull. This process ensures the path of the quadrotor remains clear of obstacles, maintaining the necessary safety margins.

4.2.2.4 Cost Function Formulation in the B-Spline Optimization

The cost function is designed to optimize the B-spline trajectory. This cost function is formulated as a sum of various components, each reflecting a specific aspect of the trajectory's quality [36]:

$$f_{\text{total}} = \lambda_1 f_s + \lambda_2 f_c + \lambda_3 (f_v + f_a) \tag{4.5}$$

Here, f_s represents the smoothness cost, f_c the collision cost, and f_v and f_a denote the costs for exceeding maximum velocity and acceleration limits. The parameters $\lambda_1, \lambda_2, \lambda_3$ are weights assigned to each cost component, enabling fine-tuning of the optimization process.

The smoothness cost, f_s, is distinct from traditional methods that often use squared snap or jerk for the smoothness cost function choice. Instead, this approach employs an elastic band cost function, which is based on the geometric arrangement of the control points. The elastic band model views the trajectory as a flexible band where each segment exerts a virtual force on its neighboring segments, pulling them towards a line that minimizes curvature and hence, smooths the trajectory [36]:

$$f_s = \sum_{i=p_b-1}^{N-p_b+1} \left\| (\mathbf{Q}_{i+1} - \mathbf{Q}_i) + (\mathbf{Q}_i - \mathbf{Q}_{i-1}) \right\|^2 \tag{4.6}$$

Here, $\mathbf{Q}_i$ are the control points of the B-spline, and p_b is the spline degree. This model effectively minimizes curvature, leading to smoother trajectories. To illustrate, the equation shows that if all terms are equal to zero, this will result in all control points forming a straight line.

Collision avoidance is handled through the collision cost f_c, which employs a differentiable potential cost function. This function measures the proximity of the spline to obstacles, and is designed to increase exponentially as the spline approaches an obstacle [36]:

$$f_c = \sum_{i=p_b}^{N-p_b} F_c(d(\mathbf{Q}_i)) \tag{4.7}$$

$$F_c(d(\mathbf{Q}_i)) = \begin{cases} (d(\mathbf{Q}_i) - d_{\text{thr}})^2 & \text{if } d(\mathbf{Q}_i) \leq d_{\text{thr}} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

Here, $d(\mathbf{Q}_i)$ is the distance from control point $\mathbf{Q}_i$ to the nearest obstacle, and d_{thr} is a threshold distance. The function F_c ensures that the optimizer increases the trajectory's clearance from obstacles.

To penalize trajectories exceeding maximum allowable velocity and acceleration, cost components f_v and f_a are defined. These are formulated as penalties that become active when the control points indicate velocity or acceleration beyond the predefined limits. The penalties for excess velocity are given by [36]:

$$f_v = \sum_{\mu \in \{x,y,z\}} \sum_{i=p_b-1}^{N-p_b} F_v(V_{i\mu}) \tag{4.9}$$

with the penalty function $F_v(V_{i\mu})$ defined as [36]:

$$F_v(v_\mu) = \begin{cases} (v_\mu^2 - v_{\text{max}}^2)^2 & \text{if } v_\mu^2 > v_{\text{max}}^2 \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

Similarly, the penalty for exceeding the maximum allowable acceleration is given by [36]:

$$f_a = \sum_{\mu \in \{x,y,z\}} \sum_{i=p_b-2}^{N-p_b} F_a(A_{i\mu}) \tag{4.11}$$

Here, v_{max} and a_{max} represent the maximum allowable velocity and acceleration, respectively. $V_{i\mu}$ and $A_{i\mu}$ denote the velocity and acceleration components in the x, y, and z directions at the control points. These formulations ensure the trajectory adheres to the specified dynamic limits by imposing squared penalties when the velocity or acceleration in any dimension exceeds these limits.

4.2.3 Time Adjustment in B-Spline Trajectory Optimization

Time adjustment is a major step of this implementation to transform the infeasible, aggressive motions that sometimes we get, even while constraining the kinodynamic search and optimization. This happens when the trajectory requires the quadcopter to cover longer distances in the same time. Time adjustment addresses this issue by adjusting the time allocation for different segments of the trajectory.

Non-uniform B-splines has variable spacing of their knot vectors. This variability allows for more control over the shape and smoothness of the trajectory. In non-uniform B-splines, the

velocity and acceleration, derived from the control points, can be expressed as [36]:

$$\mathbf{V}_i = \frac{1}{\Delta t_i}(\mathbf{Q}_{i+1} - \mathbf{Q}_i), \tag{4.12}$$

$$\mathbf{A}_i = \frac{1}{\Delta t_i^2}(\mathbf{Q}_{i+1} - 2\mathbf{Q}_i + \mathbf{Q}_{i-1}), \tag{4.13}$$

where Δt_i are the knot spans which could be modified to adjust the trajectory's dynamics in case of infeasible control points.

For a control point V_i of velocity, which is considered infeasible, we consider the largest infeasible component $V_{i\mu}$ and its magnitude $|V_{i\mu}| = v_m$. According to Equation 4.12, the velocity component is influenced by the duration between the knots. By modifying this duration, we can alter the velocity component to make it feasible [36]:

$$\hat{V}_{i\mu} = \frac{p_b}{\hat{t}_{i+p_b+1} - \hat{t}_{i+1}}(Q_{i+1\mu} - Q_{i\mu}) = \frac{1}{\mu_v} V_{i\mu} \tag{4.14}$$

Here, μ_v is set as the ratio $\frac{v_m}{v_{max}}$, ensuring that the adjusted velocity $\hat{V}_{i\mu}$ remains within the feasible limit. Similarly, for acceleration feasibility, the time span Δt_m is adjusted to $\Delta \hat{t}_m = \mu_a \Delta t_m$, ensuring that the adjusted acceleration $\hat{A}_{i\mu}$ does not exceed the maximum limit.

The process starts by identifying the control points for velocity and acceleration that are infeasible. These points are the ones where the quadrotor's capabilities are exceeded either in terms of velocity or acceleration. For each infeasible control point, the algorithm computes the necessary adjustment to the knot spans. This is done by setting the parameters μ_v and μ_a, which are then used to scale the current knot spans, as shown in Equation 4.14. The parameters μ_v and μ_a are derived based on the ratio of the maximum feasible velocity (or acceleration) to the

magnitude of the infeasible velocity (or acceleration) component. An essential aspect of this algorithm is the bounding of μ_v and μ_a with constants α_v and α_a that are slightly larger than 1. This ensures that no single time span is extended excessively, which could otherwise lead to an impractical trajectory. The algorithm proceeds iteratively, adjusting knot spans and re-evaluating the feasibility of the control points until all are within acceptable limits.

4.3 Simulation Integration with the Autopilot

4.3.1 Technical Implementation Details

The implementation of FastPlanner algorithm is made open-source[1] and is implemented using C++11 and incorporates the NLopt nonlinear optimization library.

The algorithm utilizes a Euclidean Distance Field (EDF) for environment mapping, storing distance information in a voxel grid map for collision checking and path optimization. To enhance efficiency, the map updates are localized around the drone within a predefined planning horizon to conserve computational resources.

Regarding the finer details of the implementation of the FastPlanner package [36], the kinodynamic A* algorithm, implemented in 'kinodynamic_astar.cpp', expands motion primitives, which employs a heuristic function for cost estimation to the goal. This program is responsible for collision checks and continuously updating the goal based on the drone's progression. The resultant path adheres to the user-defined maximum velocity and acceleration limits, making it dynamically feasible for the drone. Subsequently, the path undergoes refinement in 'bspline_optimizer.cpp'. The NLopt library is then used in minimizing the combined cost function, which

[1] FastPlanner Github Repo: `https://github.com/HKUST-Aerial-Robotics/Fast-Planner`

balances smoothness, distance, and feasibility.

Environment mapping and collision avoidance are managed by 'sdf_map.cpp', which establishes the EDF map. User-defined parameters such as map resolution, map size, and depth filtering are used here. Incoming depth images or point clouds are processed to update occupancy in the grid and compute the ESDF. The algorithm employs raycasting to identify occupied and unoccupied spaces. The Finite State Machine in 'kino_replan_fsm.cpp' is responsible of the planner's high-level decision-making and state management. It initializes modules based on user-defined parameters, processes new waypoints, and updates the drone's position and velocity using odometry data. It also continuously monitors for potential collisions to trigger re-planning when necessary.

4.3.2 Integration with PX4 in SITL Simulation

Integration with the PX4 autopilot stack is achieved through the utilization of ROS and MAVROS, which enables communication between the planning software and the drone's flight systems. The system integration is tested in a Gazebo simulation with a PX4-equipped drone, spawned with a model of a depth camera.

The key to adapting the planner with any ROS-based project are the parameter and launch files, particularly 'kino_algorithm.xml' and 'kino_replan.launch'. These files contain the necessary user-defined parameters and ROS topics, enabling customized planner behavior. For instance, the 'map_size_x/y/z' parameters have been tailored to match the dimensions of the operational environment. Adjustments to 'sdf_map/resolution' and 'sdf_map/obstacles_inflation' optimize the SDF map's detail level and safety buffer around obstacles. Parameters like 'max_vel' and

'max_acc' state the desired drone's physical limitations, while kinodynamic path searching and trajectory optimization parameters fine-tune the planning process. The modifications to camera and depth sensor parameters, odometry topics, and data feeds from camera and depth sensors are chosen to match the outputs from the simulated drone in Gazebo.

Fig. 4.1 shows the ROS graph for the nodes running and the topics communicated between the nodes to run the planner in SITL with PX4. The custom node 'traj_to_px4' translates Fast-Planner's trajectory commands into MAVROS compatible messages, considering coordinate frame differences and ensuring the drone interprets these commands correctly. The node continuously publishes position targets to '/mavros/setpoint_raw/local'. This communicates to the drone in the simulation, enabling responsive and fluid flight behavior based on FastPlanner's trajectory optimization and generation.

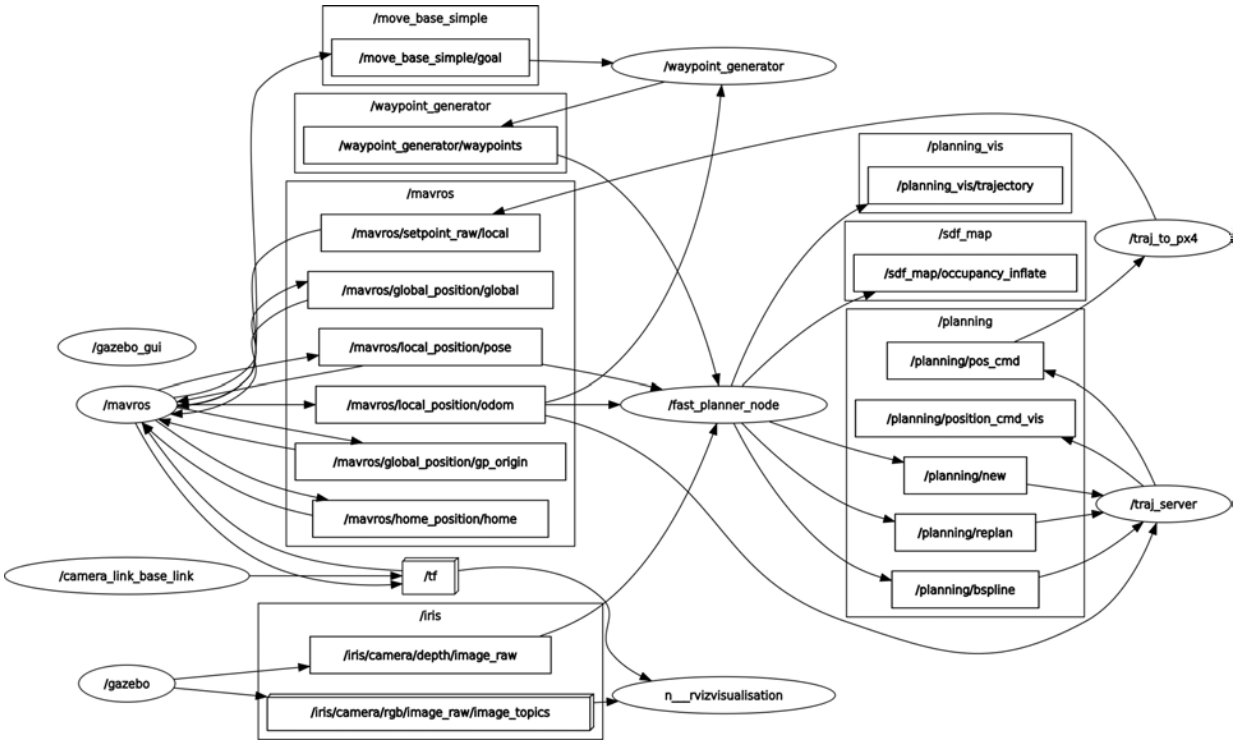


Figure 4.1: ROS graph for FastPlanner in SITL with PX4 Iris drone

Fig. 4.2 illustrates the final layout with Gazebo and Rviz. The goal is given as a 2D Nav goal in Rviz, and it sets the goal at an altitude of 2 meters, which is adjusted in the code to suit the test environment. This could be extended and changed in future improvements to enable the incorporation of varied altitudes beyond a mere 2D goal. The program then finds the path, and the custom node sends the generated trajectory as a sequence of waypoints for PX4 to follow.

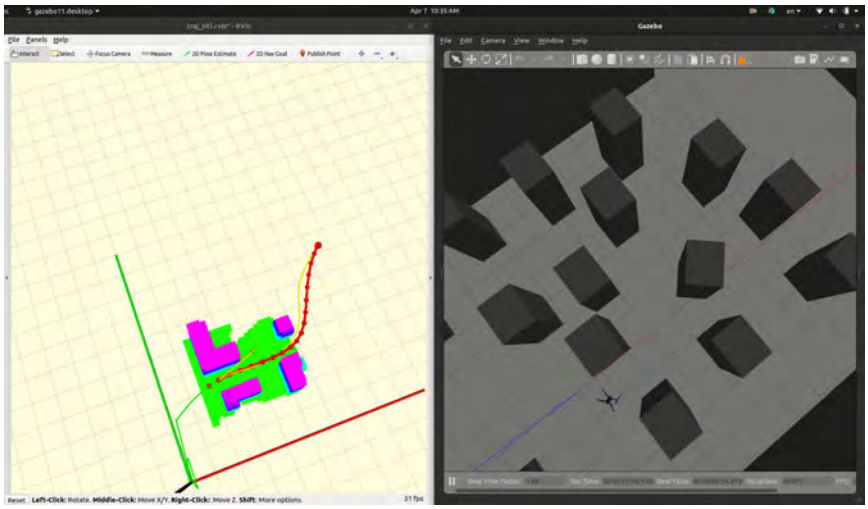


Figure 4.2: Snapshot shows the SITL Gazebo simulated environment on the right and Rviz on the left

4.4 Real-World Integration and Adaptations

4.4.1 Incorporating Depth Sensing into Drone Hardware

Moving from successful SITL testing to real-world flight necessitated augmenting our M500 drone with additional sensory capabilities. A depth sensor is necessary for building the cost map during the autonomous flight. Even though the M500 comes with stereo camera pair that might provide depth, the point cloud generated by it indoors was sparse.

To address this, we integrated a Time of Flight (ToF) depth sensor[2], specifically designed by

[2]ModalAI's Time of Flight Sensor: `https://www.modalai.com/products/voxl-dk-tof`

ModalAI for compatibility with the VOXL companion computer. The ToF sensor was selected for its high-fidelity indoor depth mapping and ability to produce dense point clouds, vital for detailed environmental understanding and accurate trajectory planning.

The choice of the ToF sensor aligns with the goal of optimizing the Size, Weight, and Power (SWaP) aspects of the drone system. Its compact size, lightweight nature, and minimal power requirements make it an ideal solution for SAR applications, where efficient use of limited computing resources is needed. The ToF sensor's performance for indoor depth mapping, with a functional range of 4-6 meters, fits with our indoor exploration objectives. Fig. 4.3 demonstrates the setup, showcasing the ToF sensor's integration with the VOXL computer and its placement on the drone, where it replaces the stereo camera.

Figure 4.3: Time of Flight (ToF) connected to VOXL m500 drone

4.4.2 Technical Integration with Onboard Sensors

Integrating ToF sensor into the M500 drone required several adjustments for full functionality. The first step involved reconfiguring the 'voxl-camera-server' to activate the ToF camera pipeline while deactivating the unused stereo camera, optimizing the processing capacity. A custom extrinsic configuration was necessary due to the ToF sensor's arbitrary mounting on the M500. Adjustments to the sensor's position and orientation relative to the drone's IMU sensor were explicitly measured and placed in the extrinsic file for accurate data interpretation. This recalibration allowed the onboard software to correctly translate and rotate sensor readings into the drone's frame of reference, aligning the sensor data with the physical world.

As for any camera sensor, the calibration process was needed in determining the intrinsic parameters of the ToF sensor. This step rectified the depth images to output precise depth mapping essential for accurate environment reconstruction and obstacle avoidance. Post-calibration, the integration of the ToF sensor with the VOXL SDK and hardware was facilitated using the 'voxl_mpa_to_ros' node in ROS. Fig. 4.4 shows the outputs as ROS topics visualized in Rviz, including aligned point cloud with the drone's pose and the corresponding depth image.

Mimicking the setup used in the SITL simulation integration, the FastPlanner node was running offboard and communicated with the drone's onboard computer using ROS over WiFi. This integration involved a network of ROS nodes to manage data transformation and transmission effectively. Fig. 4.5 illustrates a ROS graph with nodes running offboard on a local machine, and onboard on VOXL.

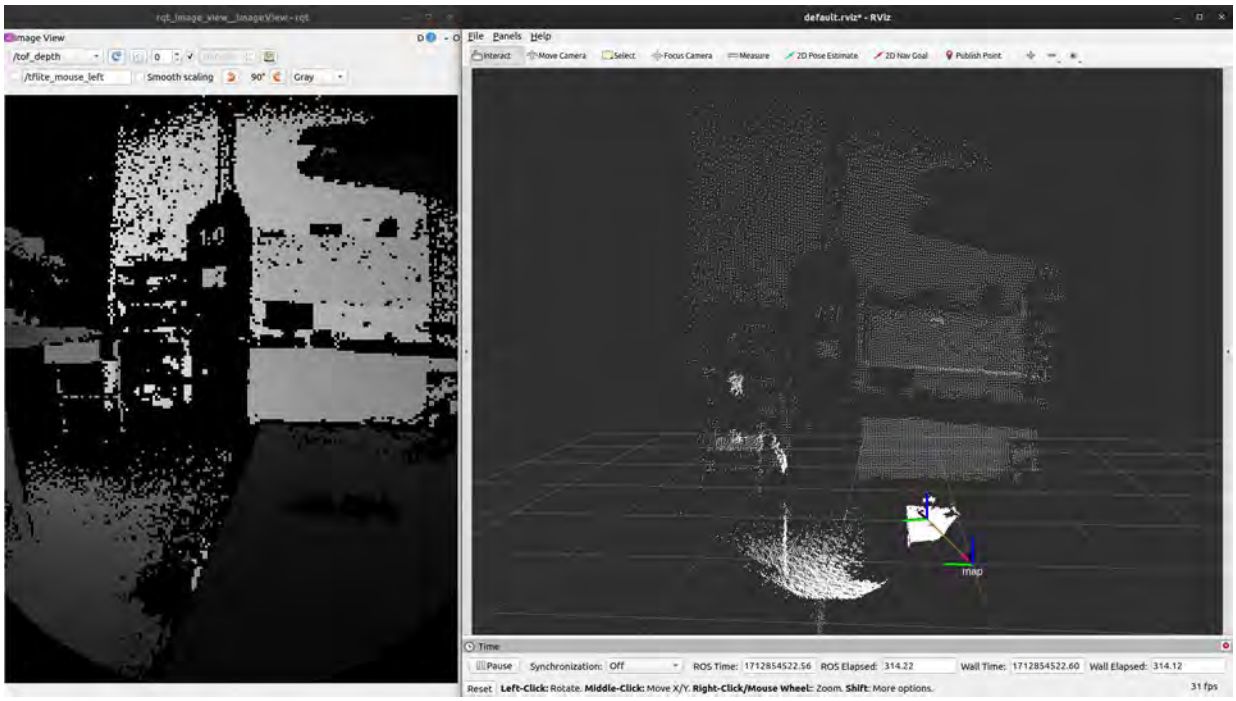


Figure 4.4: TOF outputs showing the depth image (left) and the pointcloud (right)

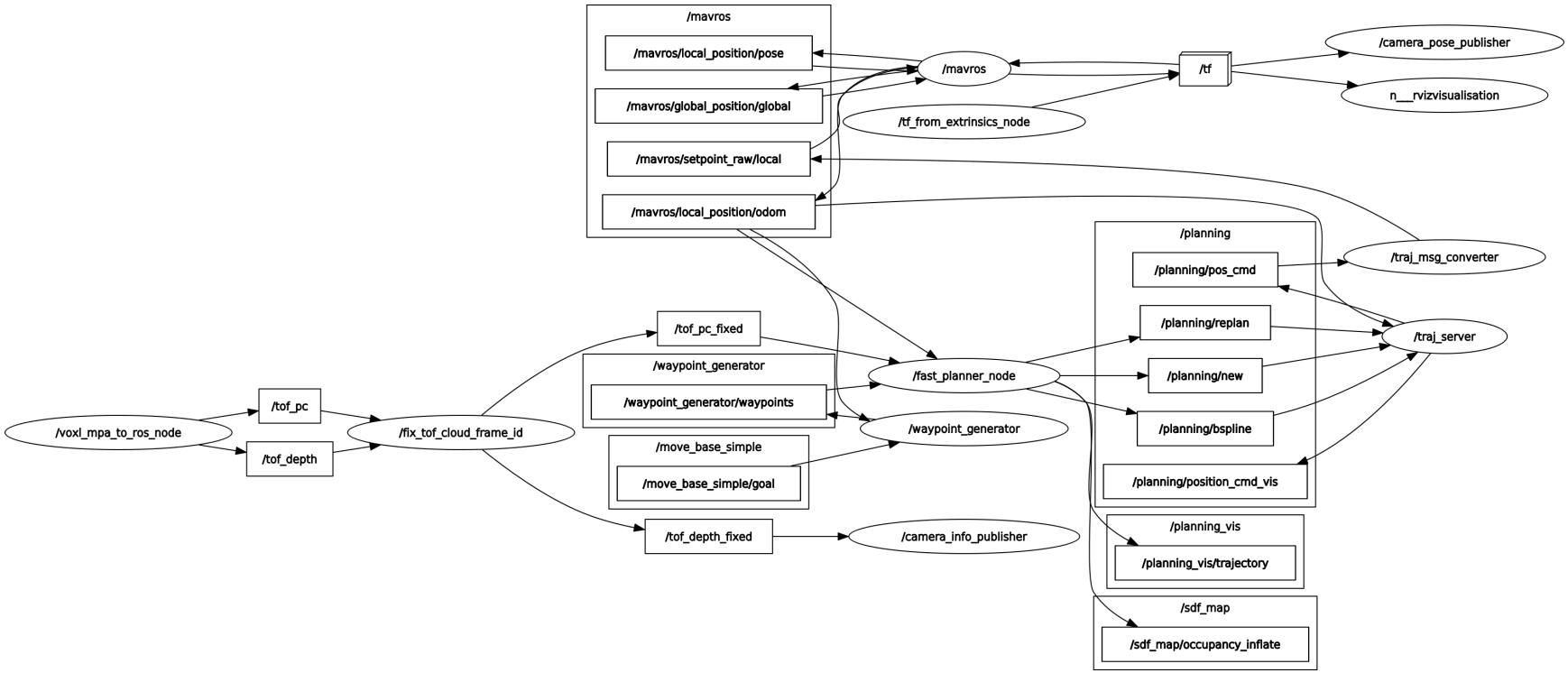


Figure 4.5: ROS graph showing nodes running to facilitate the adaptation of the planner with VOXL m500 drone

The integration workflow included several key ROS nodes:

- `camera_info_publisher` converted YAML calibration data into a CameraInfo message, providing essential calibration information.
- `image_proc_node`, part of the `image_pipeline`[3] in ROS, rectified incoming depth images to correct lens distortion using the camera info.
- `tf_from_extrinsics_node` published static transformations based on the extrinsic configuration file, ensuring accurate spatial referencing, as seen in the TF tree shown in Fig. 4.5.
- `fix_tof_cloud_frame_id` resolved frame naming discrepancies for the ToF sensor's point cloud data to properly associate the point clouds with the sensor's frame.
- `camera_pose_publisher` published the camera's pose in the map frame as a PoseStamped message.

[3]ROS image_proc Node: `https://wiki.ros.org/image_proc`

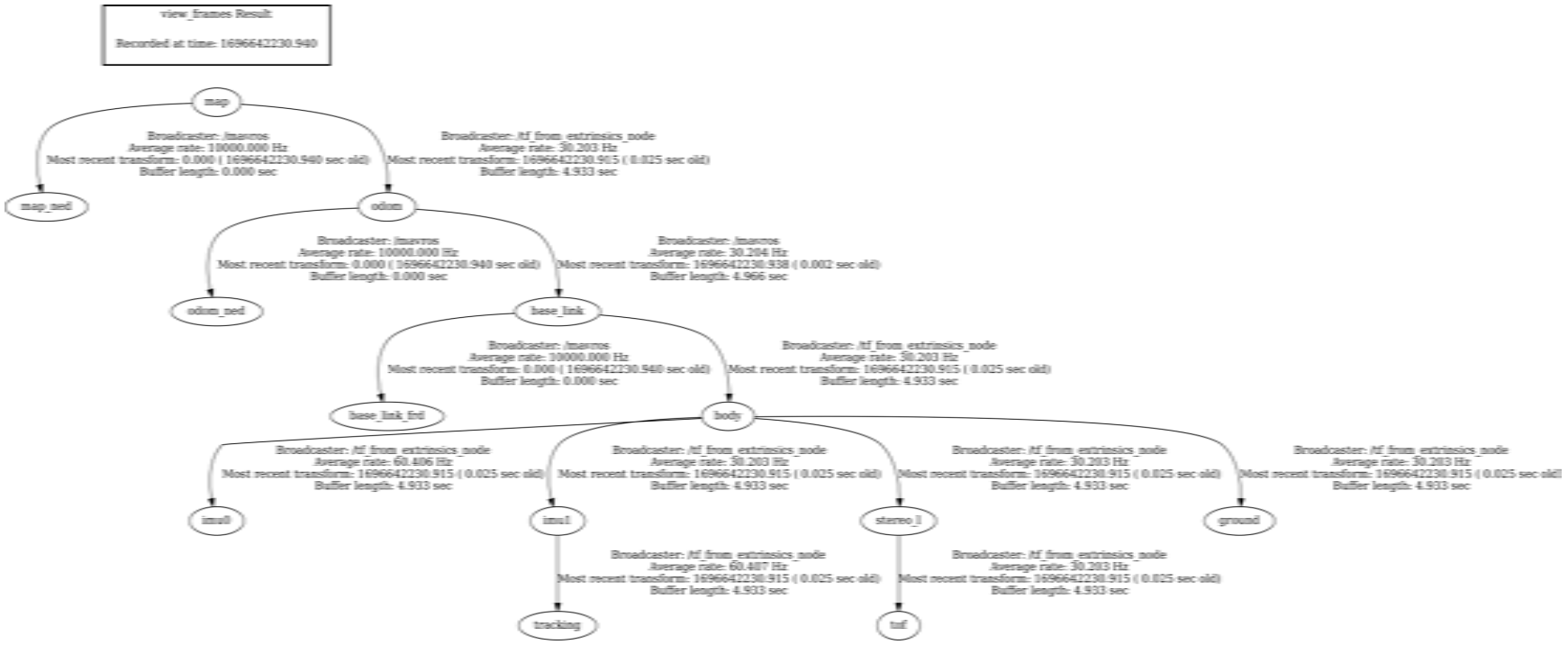


Figure 4.6: TF tree showing the different frames connected through static transformations

Instead of subscribing to simulation-based topics such as those from the Gazebo-simulated Iris drone, the planner's launch file is changed to subscribe to topics provided by the drone's onboard ROS nodes. For instance, topics like '/iris/camera/depth/image_raw' used in simulation are replaced with equivalent real sensor topics like '/tof_depth_fixed' so that the planner operates based on live sensor inputs. Additionally, some of the parameters such as the obstacle inflation have been increased to adapt for the actual size of the drone. Fig. 4.5 shows the the cost map built by FastPlanner using the ToF sensor readings from a real experimental setup.

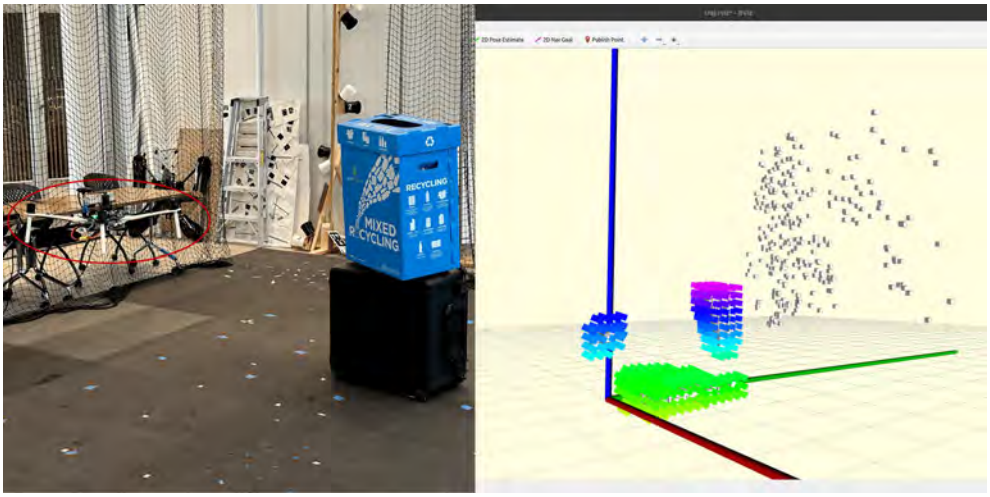


Figure 4.7: Depth mapping from real ToF readings during hover with one obstacle in front of the drone

4.4.3 Challenges and Proposed Solutions

4.4.3.1 Sensor Data Processing

Some challenges emerged when testing the planner experimentally on the drone. Firstly, the map generated from the ToF point cloud seems to be subject to noise, where some inconsistent point clouds confuse the algorithm and could be seen as spurious obstacles in the environment. Given that FastPlanner's original implementation has been done with Light Detection and Ranging

(LiDAR) sensor, its processing algorithms might not effectively filter out ToF noise, leading to a cluttered and inaccurate representation of the surroundings in the planner's spatial map as shown in Fig. 4.8. Another reason could be the arbitrary mounting with of the ToF sensor with no damping to cancel out body vibrations that might affect the accuracy of the point cloud.

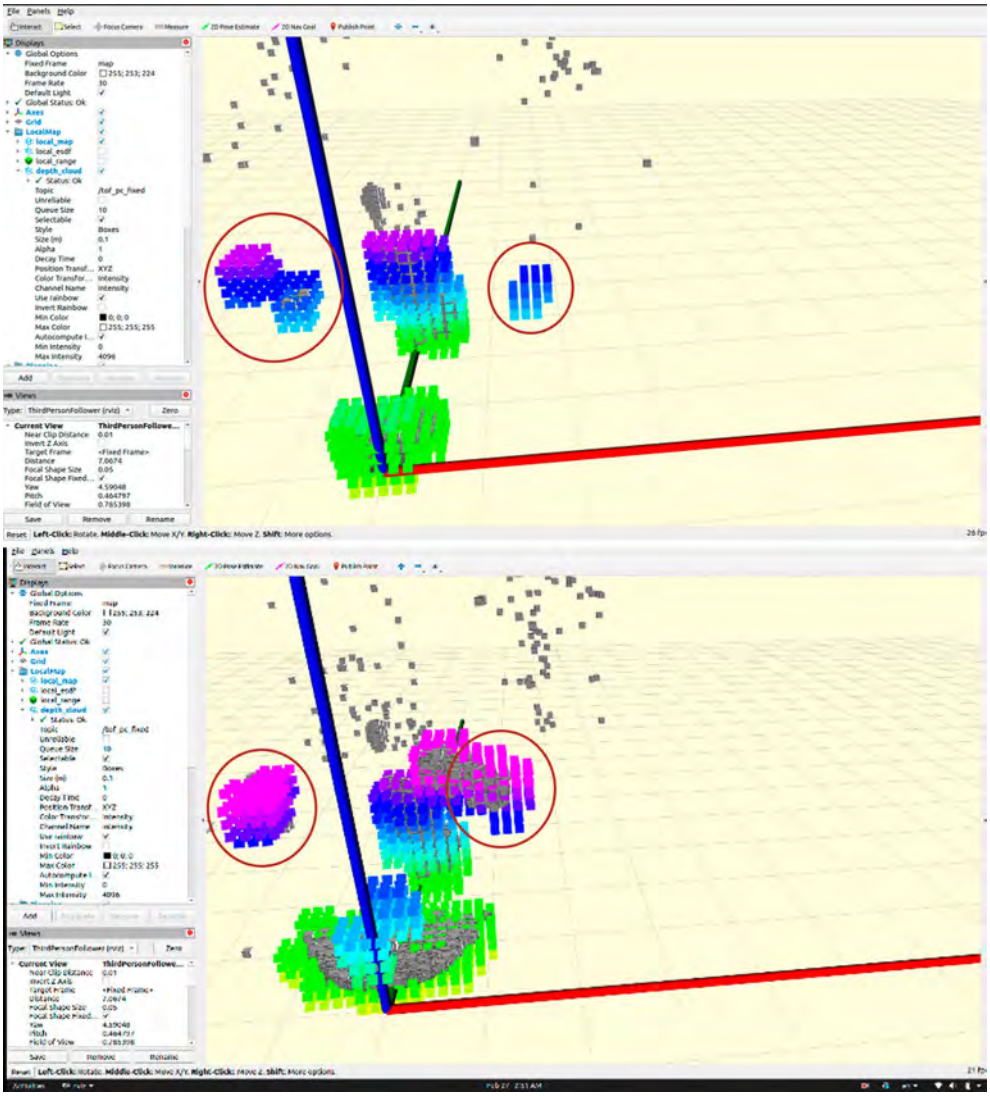


Figure 4.8: Noisy point cloud causes noisy cost map during the drone's movement

4.4.3.2 Latency in Offboard Control

Secondly, there is an inherent latency in the offboard control setup that we use, where the planner is running on a local machine which must receive, process, and respond to data transmitted via WiFi over ROS. Each step in this pipeline, from data transmission to processing on the local machine and back to the drone, introduces delays. These delays mean the planner often works with outdated information and it leads to inconsistent behaviour.

4.4.3.3 Proposed Solutions

Starting with enhancing the processing of sensor data, given the ToF sensor's susceptibility to noise, the planner algorithm could be adapted to better handle this type of data. Implementing advanced noise reduction techniques tailored to ToF sensors, such as dynamic thresholding would be a proposed solution. Additionally, integrating sensor fusion techniques that combine ToF data with other sensory inputs could provide a more comprehensive environmental understanding, and better mapping.

To tackle the issue of latency in offboard processing, an optimal solution is to shift the data processing onboard the drone's companion computer. Doing so would reduce the volume of data transmitted and processed offboard, thereby diminishing latency. This will require upgrading the current companion computer; VOXL to a more powerful processor which can handle such a load of computation running at once. One natural upgrade is the VOXL2 [4], which is developed by the same company, and is more powerful than VOXL1.

In terms of path planning, fine-tuning the planner's parameters like the obstacle inflation

[4]VOXL2 Companion Computer: `https://www.modalai.com/products/voxl-2`

factor and path search settings to suit the real drone environment and the ToF sensor capabilities is necessary. This fine-tuning process should be continuous, utilizing empirical data gathered during flight tests to incrementally improve the planner's performance. Moreover, while the drone's yaw is considered for path alignment, it appears that the optimization of yaw movements within the trajectory planning is not fully refined. This is evident from observations in simulation where the drone sometimes executes abrupt turns or does not take the most efficient yaw turn. To enhance the planner's performance, a potential improvement would involve a deeper integration of yaw behavior into the cost function used for trajectory optimization. By incorporating terms that specifically penalize abrupt or inefficient yaw changes, the algorithm can be tailored to optimize not just the path but also the rate and direction of orientation changes.

Chapter 5: Evaluation and Results

5.1 Analysis of Autonomous Search, Detection, and Inspection

In Chapter 3, we focused on developing a framework for autonomous search, object detection, and inspection using drones in indoor environments. The methodologies implemented are now evaluated against their intended objectives. In this section, we'll dissect the outcomes of these implementations, particularly their operational efficacy as observed in experimental settings.

Autonomous Search Routine Evaluation

Fig.5.1 shows the autonomous search routine, where the drone was programmed to execute a search pattern over a defined area measuring 3.6m by 4.8m at a constant altitude of 1.1m. This figure illustrates three paths: the ground truth path as recorded by the motion capture system, the onboard estimate of the drone's trajectory, and the planned path designed to cover the search area.

Notably, the drone's actual path demonstrates a slight deviation from the planned trajectory over time. This deviation is one of the known issue of drift in visual inertial odometry-based navigation. As the drone navigates the space, accumulating positional data over time, this drift

becomes more obvious. The drone's internal estimates remain closely aligned with the planned path, staying always within the acceptable boundaries of the designated search area.

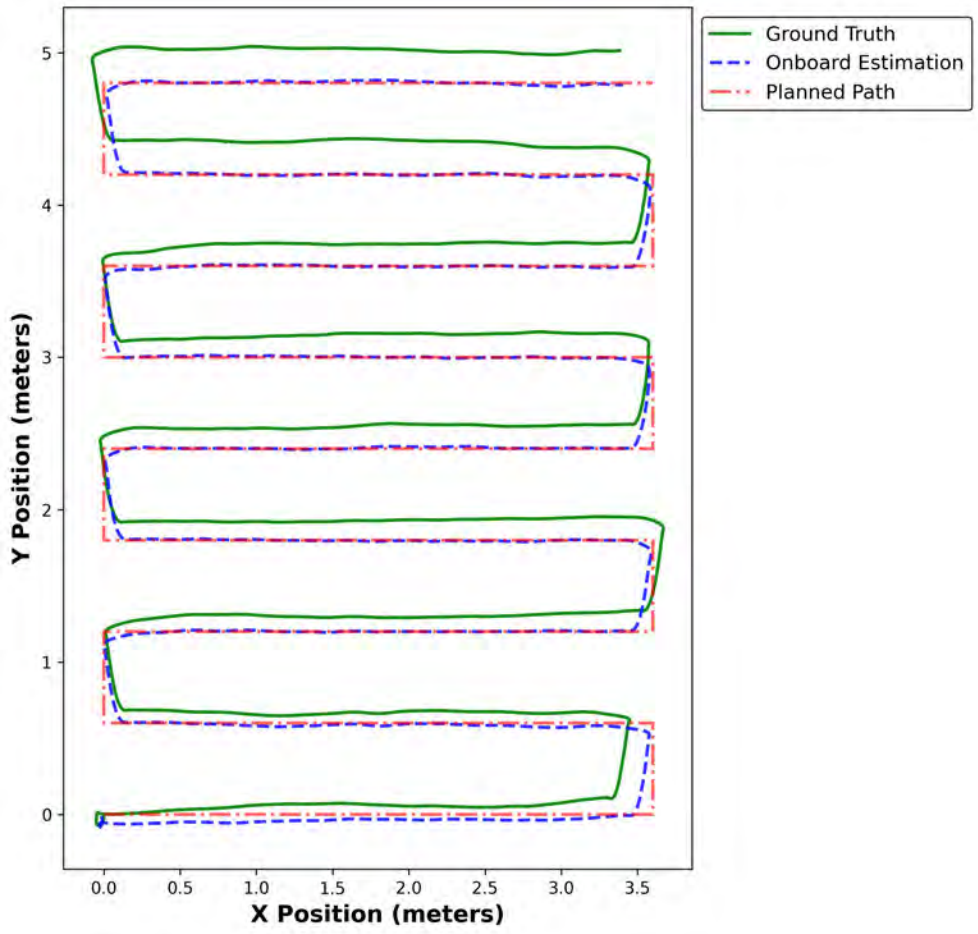


Figure 5.1: Comparison of drone's ground truth, onboard estimated, and planned search paths

While the drift observed is relatively minor and does not significantly impact the overall mission's success in this confined test environment, it could be enhanced by improving the state estimation algorithm indoors and integrate Simultaneous Localization and Mapping (SLAM) algorithms to reduce cumulative errors over time.

AprilTag Pose Estimation Evaluation and Impact on Target Localization

The search routine is helping the drone to localize the targets in the area by looking for their corresponding fiducial markers. Hence, the AprilTag pose estimation's accuracy is crucial for the drone's target localization and the subsequent inspection routine. During experiments, the drone's movement, combined with motion blur and varying lighting conditions, initially created challenges in accurately determining the marker's pose from the first detection. This is evident from Fig. 5.2, where the drone's first estimation of the tag's position showed a discrepancy of approximately 5cm from the ground truth. However, implementing a pause in the drone's movement to allow for hovering significantly enhanced the accuracy of the pose estimate. This refinement is significant, as the inspection routine's effectiveness is directly tied to the precision of AprilTag localization. This adjustment, while introducing a slight delay in the operation, results in a more reliable estimate, bringing it closer to the ground truth.

Analysis of Detection Confidence in Relation to Inspection Time

To further illustrate the relationship between the quality of the tag's localization and the inspection routine, Fig.5.3 showcases the detection confidence plotted against inspection time for different buckets of Target 0 (T0[1]). Here, we see the detection process reporting, one at a time, the bounding box coordinates in the image frame along with the confidence probability of the detected class.

As detailed in Section 3.5, the drone autonomously navigates to inspection poses upon locating the corresponding AprilTag. A key factor governing the transition from one inspection

[1]Throughout the remainder of the results, 'T' will denote 'Target' (e.g., Target 1 as T1) and 'B' will denote 'bucket' (e.g., bucket 2a as B2a). For further details on each target's specifics, see Section 3.2.1.

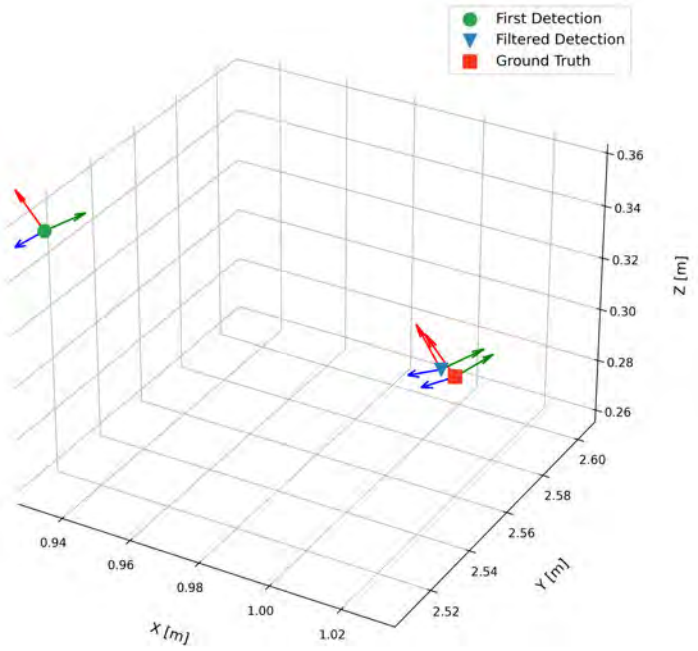


Figure 5.2: 3D pose estimation comparison between the first estimate, the filtered estimate, and the ground truth pose of the tag

point to the next is the program's confidence in the current bucket's detection. If the drone stays looking at a bucket for the maximum allowed time without achieving the acceptance threshold for detection confidence, it reports the bucket as an anomaly. In the mission reported in Fig. 5.3, the acceptance threshold for the average confidence was set at 85% as shown in the dashed line, with a minimum required number of readings set at 50. Considering the object detection model reports inferences at approximately 10Hz, it takes around 5 to 6 seconds to gather the necessary amount of readings to evaluate against the confidence threshold. The maximum wait time at each bucket is set to 20 seconds.

Fig. 5.3a shows this during inspection windows across the 8 buckets, where detection confidence consistently exceeds the 85% threshold. This resulted in an even time allocation for each inspection window, averaging around 14 seconds. This timing includes approximately 5-6 seconds of inspection per bucket, followed by an orientation change to inspect the next bucket,

and navigation to the next set of buckets for the proceeding inspection window.

Conversely, Fig. 5.3b highlights instances where the drone spent the maximum time allowed at specific markers, particularly buckets 2 and 2a, which were then reported as anomalies. The initial localization of the targets notably influenced the inspection window for these two buckets. Their pre-recorded inspection required close proximity to the AprilTag, which in turn required precise localization for clear detection, explaining their longer inspection time compared to other markers.

Fig. 5.4 further illustrates this issue in action, contrasting the drone's inspection positioning for bucket 2 with filtered (a) and initial (b) tag pose estimations. The precision of localization directly impacts the inspection's effectiveness; the filtered estimate leads to a more centered and clear view of the bucket, whereas the initial estimate results in a less focused and thus, lower confidence inspection. This comparison underscores the significance of the filtering process in enhancing the accuracy of the inspection routine and highlights the need for precision in the drone's target localization mechanism to integrate dynamic corrections for the target's location while executing the inspections.

Fig. 5.5 shows the autonomous inspection routine for T1 in action. The left side of the figure displays the drone's actual orientation during the flight, while the right side shows the output of the onboard object detection model. The figure demos how the drone navigates through specific inspection points, systematically focusing on each bucket. Initially, the drone starts centering B3, then it yaws to inspect B3a, then it progresses to B4 and B4a according to the pre-recorded inspection poses. During this phase, the model's output displays the detection bounding boxes, providing real-time feedback on the detection process.

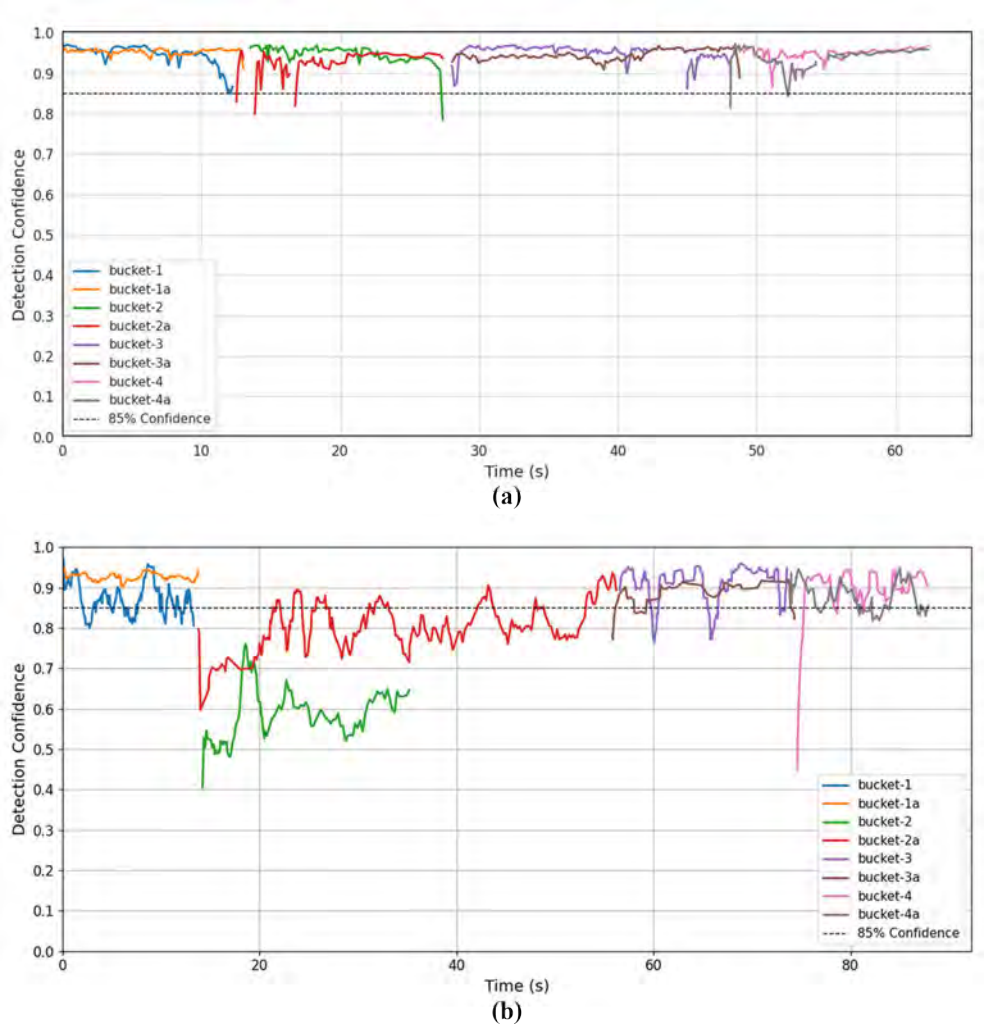


Figure 5.3: Detection confidence vs. inspection time for Target 0 when using AprilTag pose: (a) after filtering, and (b) using the initial estimate

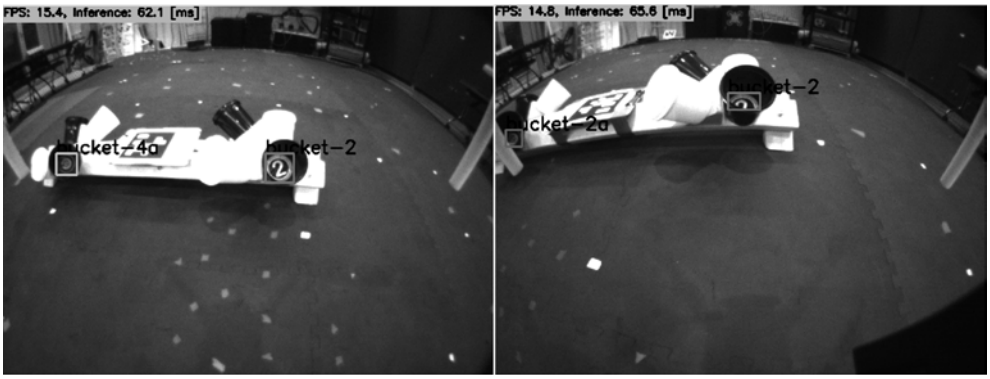


Figure 5.4: Comparison of experimental drone inspection positioning using filtered (a) and intial (b) AprilTag pose estimations

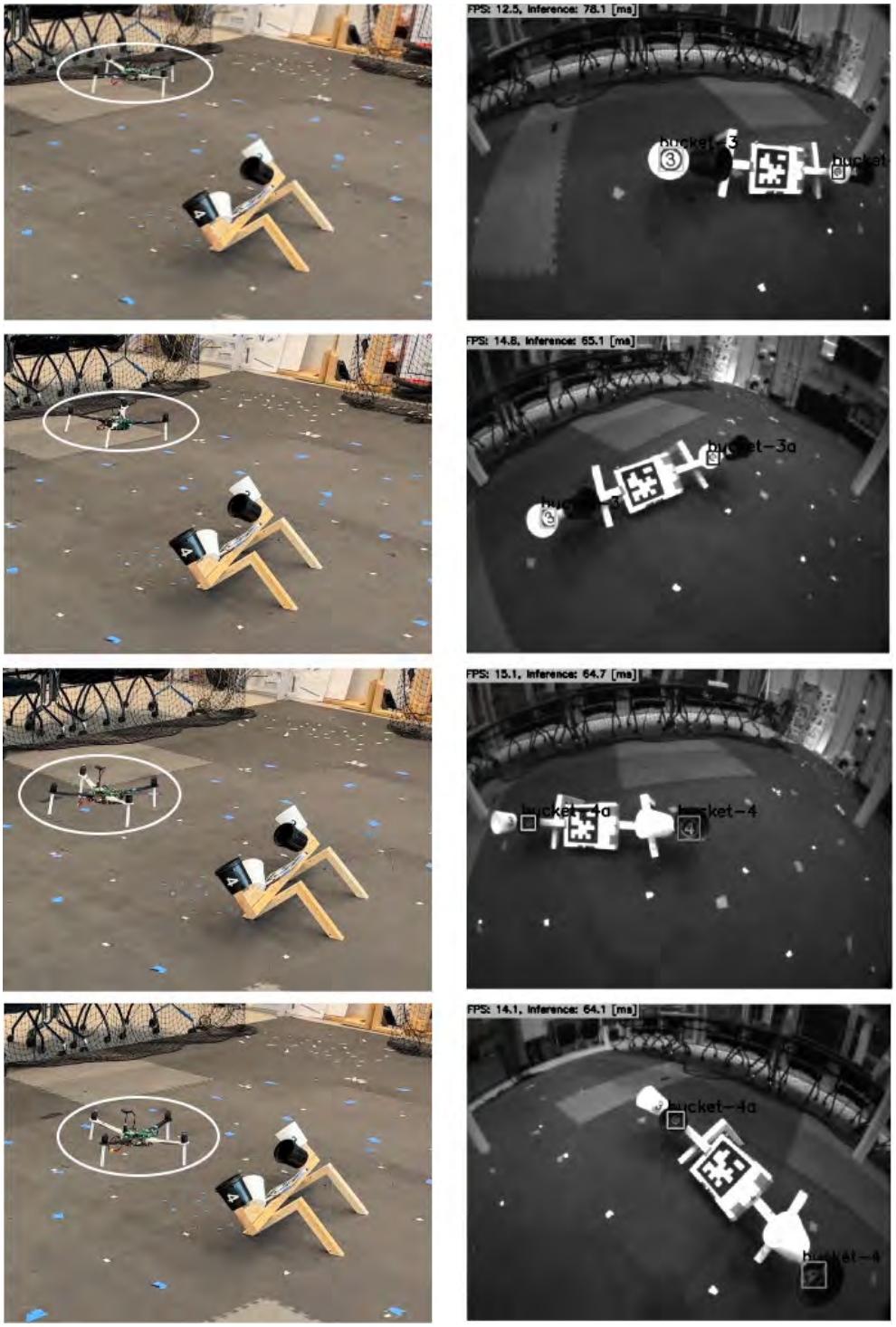


Figure 5.5: Illustration of the drone's autonomous inspection routine execution for T1, showing drone positioning and real-time object detection outputs for each marker

Field Evaluation of Full System Integration

To evaluate the system integration for the full mission, we test that using by placing the three targets randomly in the lab for the drone to autonomously search for them and inspect them. Fig. 5.6 presents illustration of the system's integrated functionality, where it shows the drone's actual trajectory in the lab's netted area from the real experiment, paired with a 3D plot that showcases the interplay between search and inspection phases. The 3D trajectory visualizes the drone's systematic transition between searching for targets and shifting into inspection mode upon target localization. The smoothness in this transition back and forth underscores the efficiency of the integrated system.

Furthermore, Fig. 5.7 incorporates a heatmap plot, which points out areas where the drone spent the most time during the mission. This heatmap could be looked at as an indirect indicator of potential anomalies, providing insights into mission execution. Such insights can guide future mission planning, which would enable more efficient path planning and resource allocation during actual deployments in SAR missions.

Fig. 5.8 presents a series of mission scenarios, each featuring different arrangements of targets within the indoor area. This diversity in target placement provides a way to assess how the drone adapts its search and inspection strategies in varied cases. The figure maps the drone's 2D path in each scenario as a visual representation of the autonomous mission executed. Table 5.1 quantifies the outcomes of each mission. It shows the details about the total mission time, inspection duration for detected targets, the count of accurately detected markers inside the buckets, and the reported potential anomalies. 'Markers Detected' reflects accurately identified labels out of the total present, confirmed with average confidence $\geq 85\%$. 'Anomalies Identified'

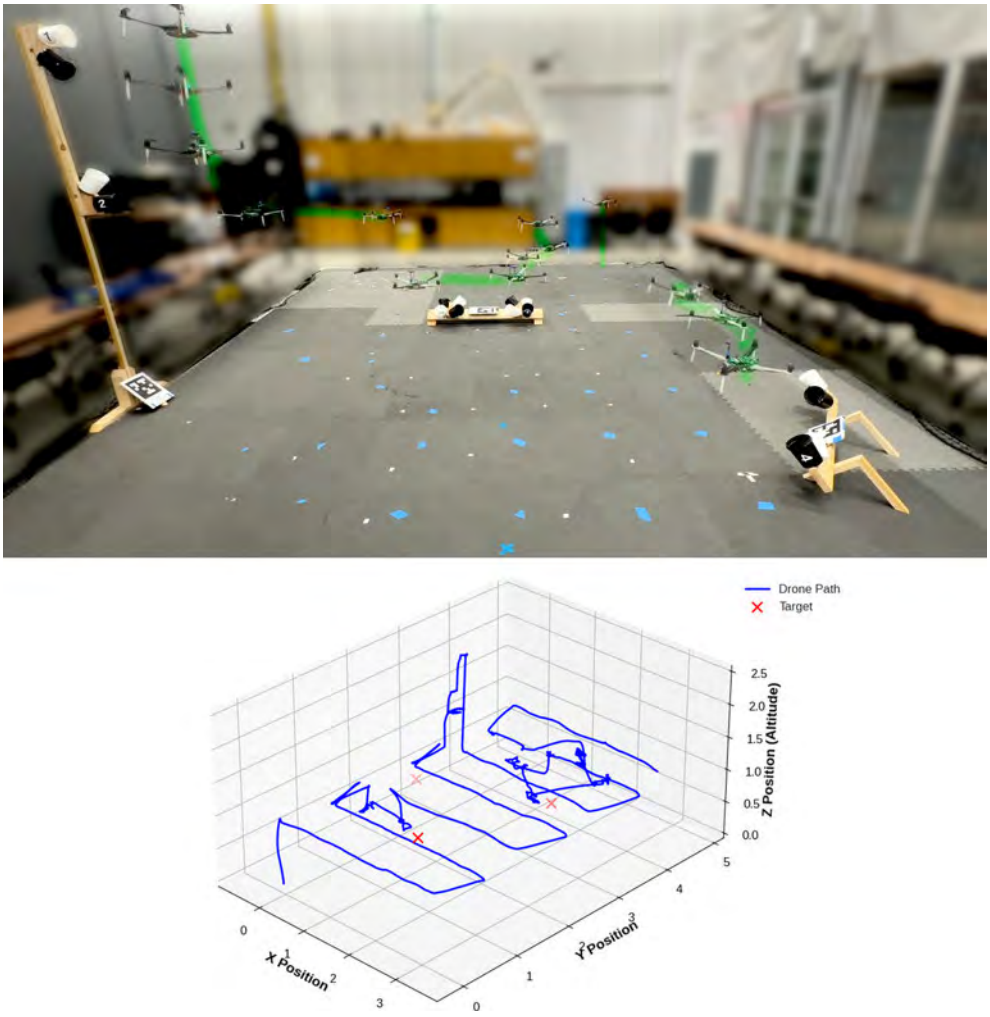


Figure 5.6: 3D representation of the drone's complete mission trajectory in the experiment, showing the transition between search and inspection phases

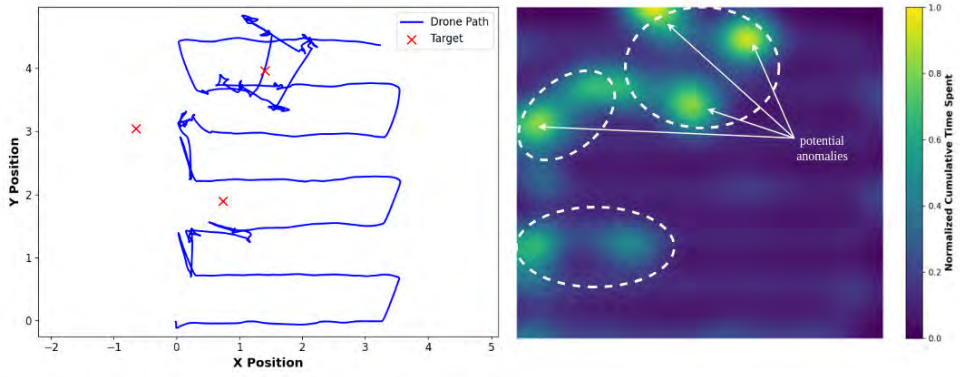


Figure 5.7: 2D path and corresponding heatmap, highlighting time spent in various mission areas and identifying potential anomalies

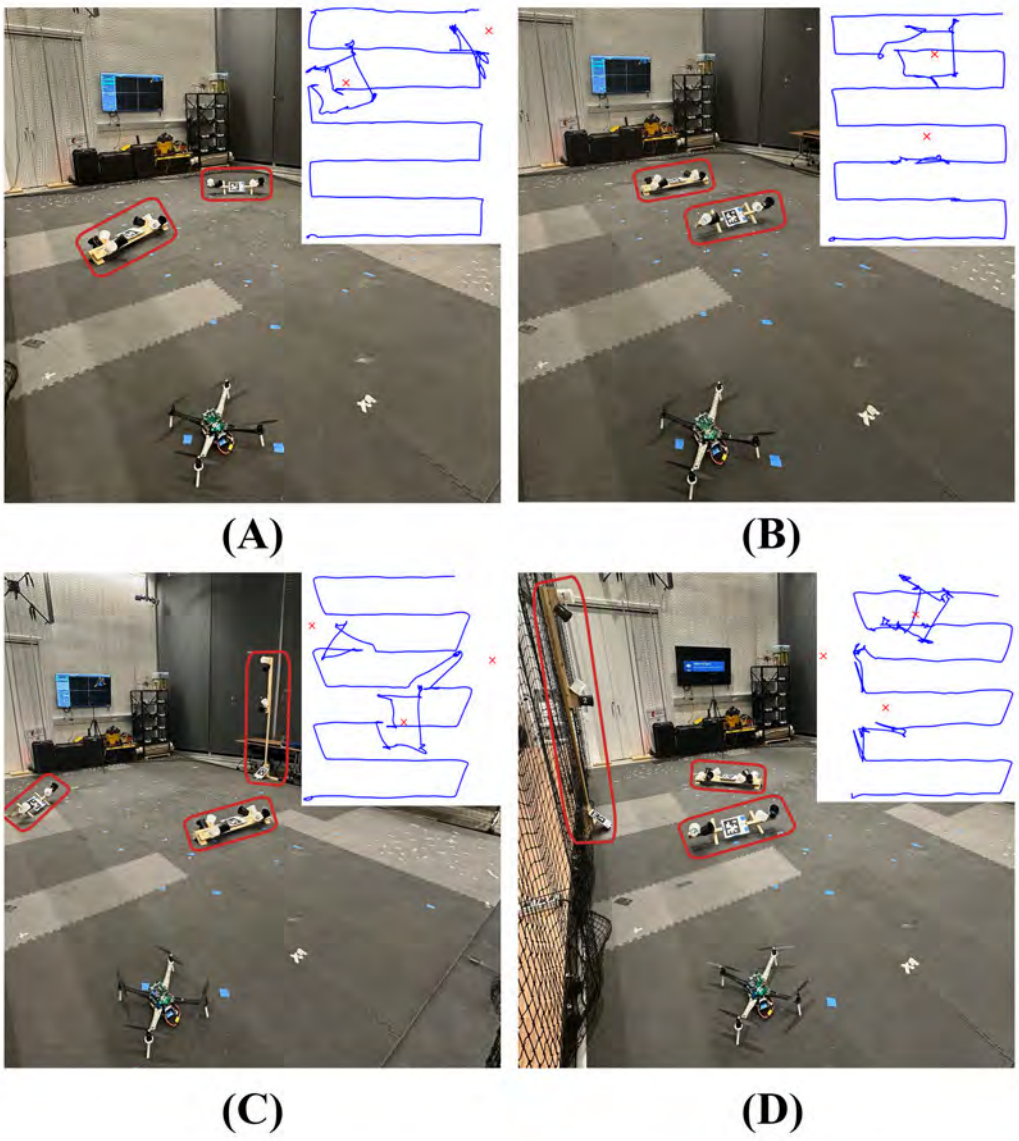


Figure 5.8: Drone's 2D path across various mission scenarios with different target placements in the experiments

Table 5.1: Tabulated results depicting inspection times, target detection efficiency, and anomalies identified for Targets 0 to 2 (T0-T2), across multiple missions, demonstrating the system's operational capabilities and areas for optimization.

Mission ID	Mission Time	Target 0 (T0)		Target 1 (T1)		Target 2 (T2)		Markers Detected	Anomalies Identified
		Inspection Time	Mean	Inspection Time	Mean	Inspection Time	Mean		
A	317.53 s (5.29 min)	94.88 s (1.58 min)	95.93 s (1.60 min)	54.87 s (0.91 min)	40.89 s (0.68 min)	—	28.54 s (0.48 min)	9/12 (75%)	3 (B2 at T0, B4 & B4a at T1)
B	302.34 s (5.04 min)	82.70 s (1.38 min)		34.65 s (0.58 min)		—		12/12 (100%)	0
C	322.35 s (5.37 min)	92.54 s (1.54 min)		40.31 s (0.67 min)		29.48 s (0.49 min		14/15 (93.3%)	1 (B2 at T0)
D	330.37 s (5.51 min)	113.61 s (1.89 min)		33.72 s (0.56 min)		27.60 s (0.46 min)		13/15 (86.7%)	2 (B2 & B4 at T0)

enumerates buckets not meeting the confidence threshold, with 'B' indicating 'bucket'.

Target arrangement influences mission outcomes. Aligned targets in Mission B, unlike in Mission A, led to reduced operational times and zero anomalies, hinting at environmental effects on system performance. The consistent detection accuracy across scenarios underscores the detection model's robustness. Yet, longer inspection times in some missions suggest the potential for system refinement.

The table reveals the effect of target orientation on inspection times. For T0, Mission B was anomaly-free, with a total inspection time of 82.7 seconds. Conversely, Missions A and C each reported an anomaly in bucket B2, and Mission D reported two anomalies, extending the inspection duration to 113.61 seconds. T1 required more inspection time in Mission A, resulting in two anomalies, while other missions reported none. Inspection times for T2 in Missions C and D were comparable, with no anomalies noted. Target positioning provides additional context. Missions A and B, hosting the same targets but with varying inclinations, demonstrate how

shadows affect detection—darker buckets like B2, B2a, B4, and B4a prove more challenging due to diminished visibility in shadowed areas.

5.2 Performance in Simulated Cluttered Environments

The results presented in this section are derived from the SITL simulation using a Gazebo environment, where an Iris drone, augmented with a depth sensor, executes a mission under the control of a PX4 flight controller. The planner operates in offboard mode, receiving commands from the ROS environment visualized in Rviz on the left side as shown in Fig. 5.9.

The figure shows the drone's environment as sensed by the depth sensor, which informs the planner's decision-making process. The ESDF voxel grid occupancy map visualized in Rviz reflects the environment's three-dimensional structure. In the planning sequence, the red trajectory indicates the original planned path, derived from the existing knowledge of the environment. This path is then further optimized, marked in yellow. These two trajectories adapt to the drone's live positional feedback and environmental data. The green trajectory showcases the drone's actual flown path. When setting a new goal through Rviz's 2D nav goal feature, the planner adjusts the drone's horizontal orientation towards the goal, while maintaining altitude consistent with the drone's current flight level. Future implementation could be made to allow for three-dimensional navigation goals.

The figure also shows how re-planning occurs in response to the drone's detection of new obstacles while navigating through unknown space. This is evident in the shift from the second to the third image, where the drone's path adapts to the newly perceived obstacles.

Fig. 5.10 presents the local planning strategy executed within a defined receding horizon.

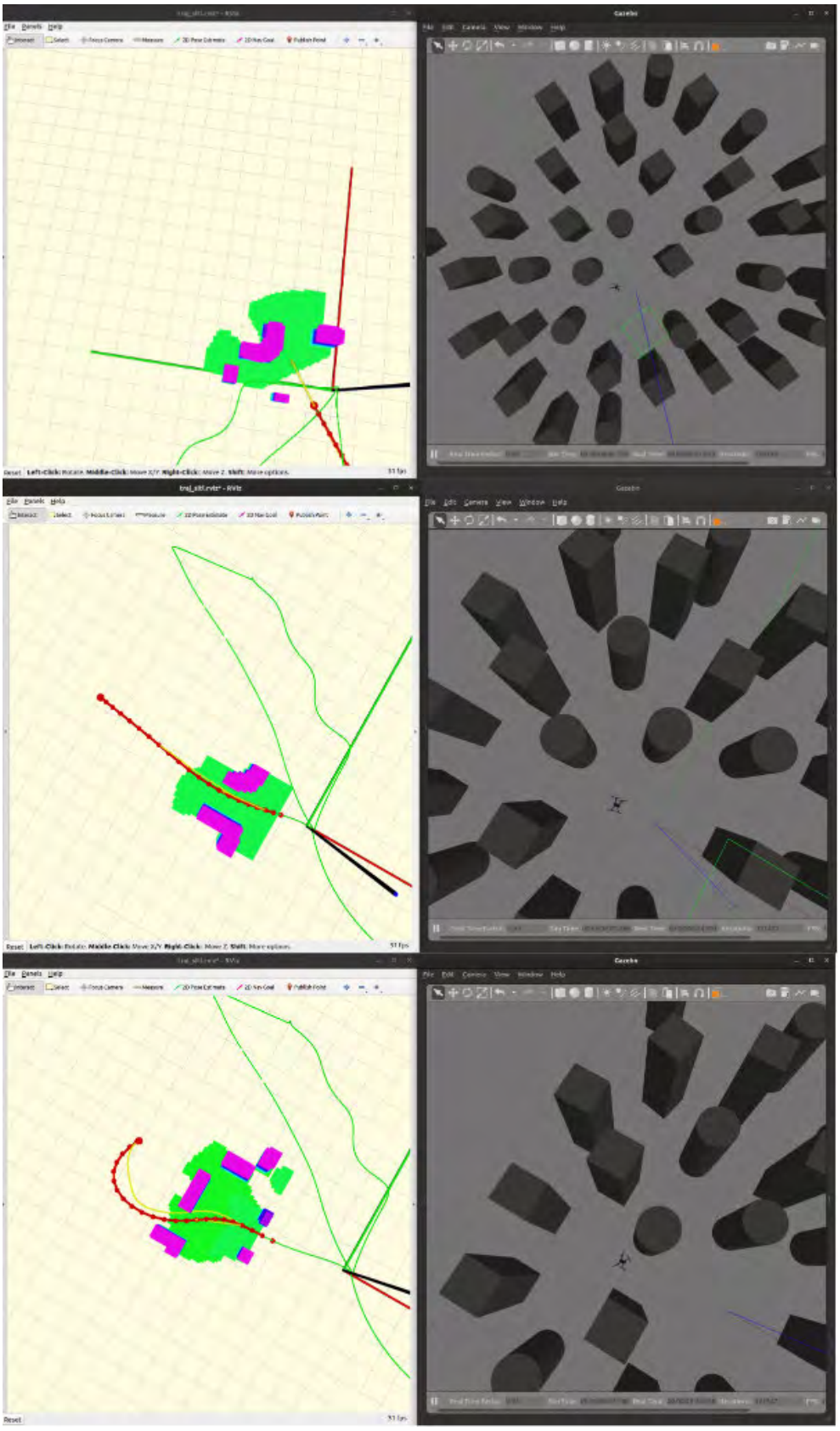


Figure 5.9: Visualization of autonomous drone path planning in SITL simulation environment

The approach constrains map generation and pathfinding to the immediate area defined by the user around the drone, as visualized in the first image via Rviz. This technique conservatively utilizes computational resources by only processing the necessary spatial information, thereby negating the need to build and frequently update a larger, more complex map.

The second image in Fig. 5.10 demonstrates the planner's proficiency in navigating within the local horizon. Starting from the origin point, the algorithm successfully computes a path that guides the drone through the constraints of the simulated environment. The results underscore the planner's ability to work with limited data to deliver a viable navigation path, underscoring its potential utility in real-world applications where processing power and time are at a premium.

Fig. 5.11 highlights the critical role of parameter tuning in the planning algorithm, specifically regarding obstacle inflation and the collision cost's weighting factor within the cost function. The top image reveals the consequences of an inadequate obstacle inflation value, which neglects to accommodate the drone's physical dimensions. This results in the generation of a seemingly feasible but ultimately unsafe path, leading the drone to erroneously attempt passage between closely spaced obstacles, resulting in collision.

In contrast, the bottom image demonstrates the impact of a better value for the inflation coupled with a higher weighting factor for the collision cost. These adjustments produce a slightly more conservative map where obstacles are appropriately inflated, thereby preventing the planner from proposing a path between the narrowly spaced obstacles.

Fig.5.12 illustrates a scenario that reveals a limitation of the local planning horizon in the context of large obstacles, such as walls. The depicted scenario reveals that the planner suggests a direct trajectory towards the goal, despite it being located on the other side of a wall, resulting from the goal's position extending beyond the local horizon of the map. This behavior is not

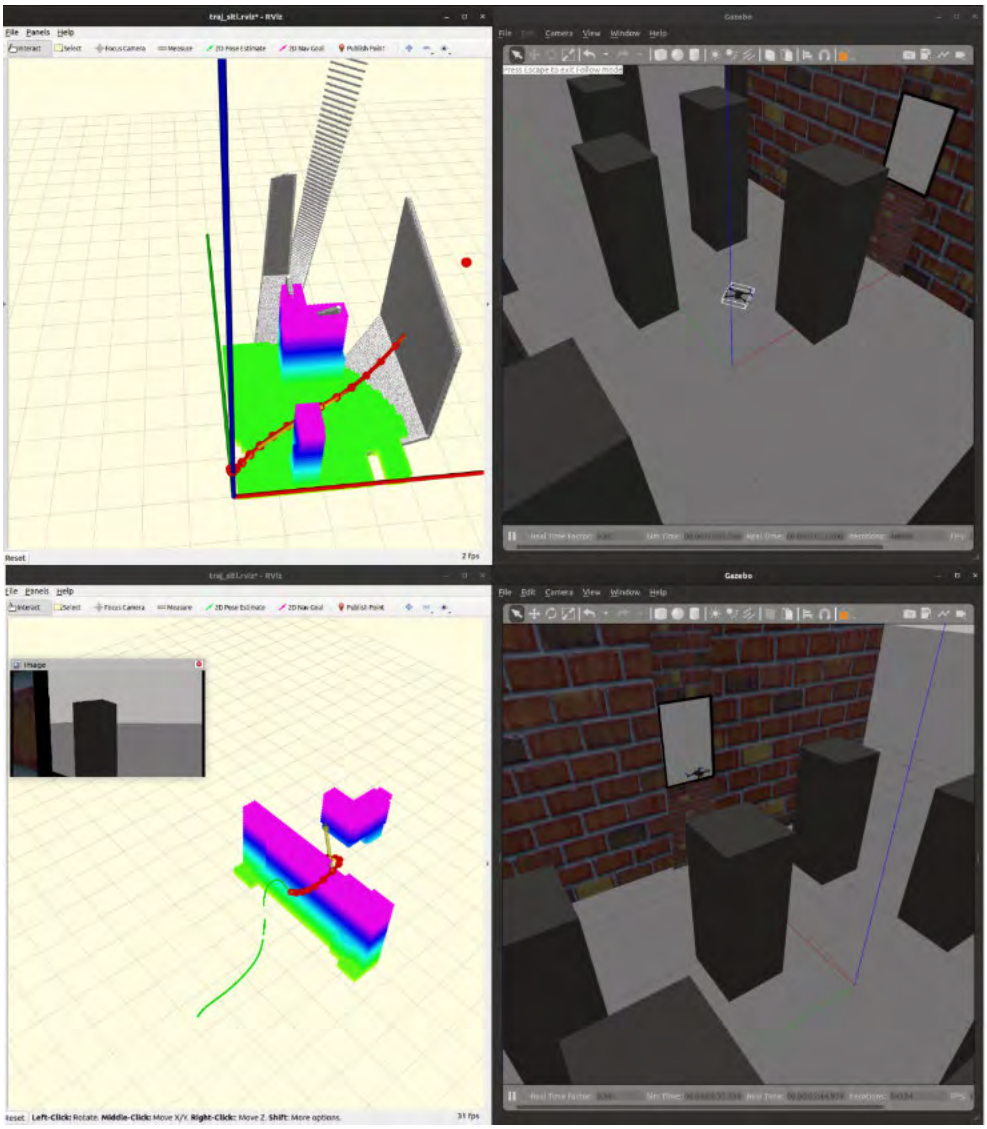


Figure 5.10: Local planning horizon implementation in drone trajectory planning for computational efficiency

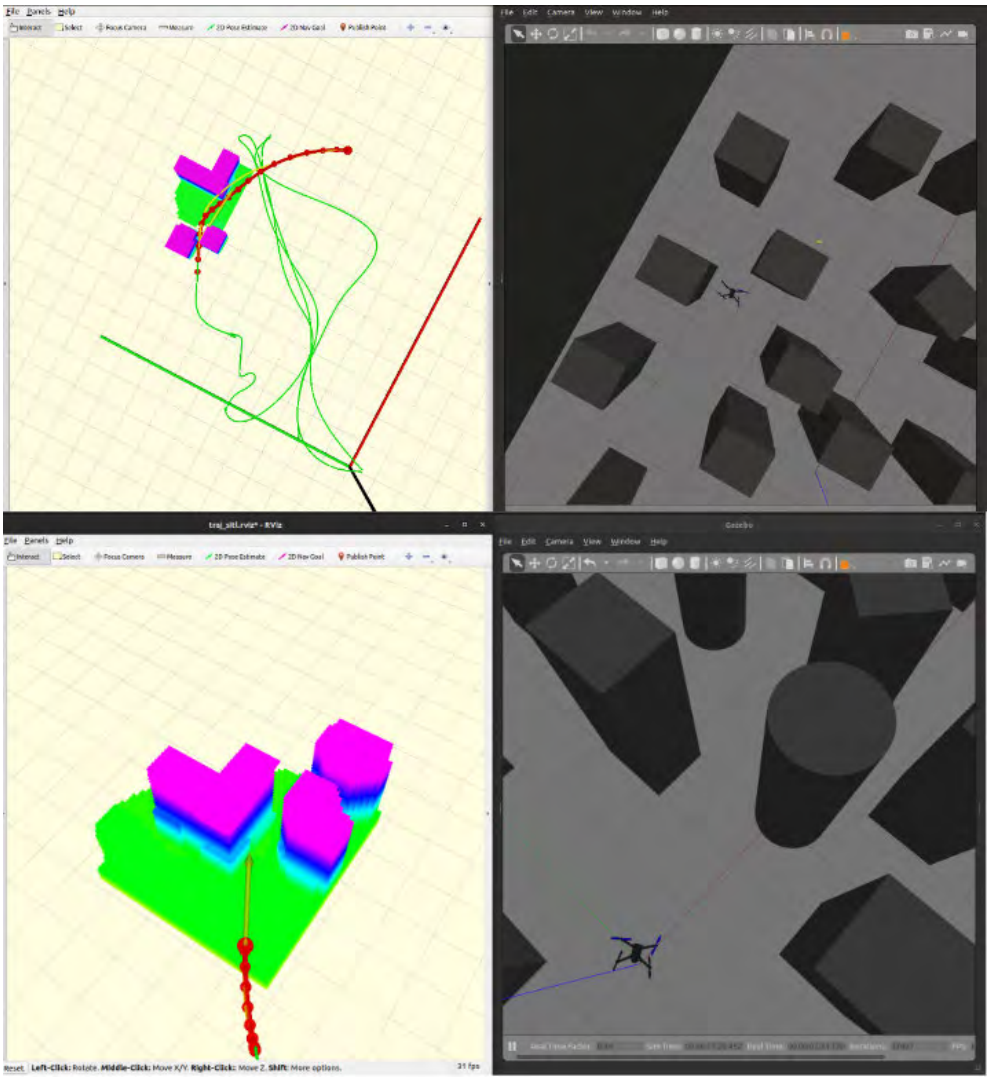


Figure 5.11: Effect of parameter tuning on drone path planning: obstacle inflation and collision cost

ideal, as it prompts the drone to attempt to navigate through the wall, an obviously infeasible action. In practical terms, the drone's actions become inefficient, characterized by a back-and-forth motion along the wall as it continuously engages in re-planning to find a viable path around the obstacle. This process is time-consuming and may not lead to the most efficient route.

Moreover, since the yaw angle is not factored into the cost function of the planning algorithm, the drone may execute unnecessary rotations to align with the desired orientation, even during the search for a path while in motion. This leads to a suboptimal behaviour in terms of energy, which could be minimized if energy efficiency was a considered component in the planning cost function.

In conclusion, although the planner effectively navigates within its local scope in near real-time, there's room for improvement. Specifically, it needs enhancements to manage larger obstacles better and incorporate energy efficiency.

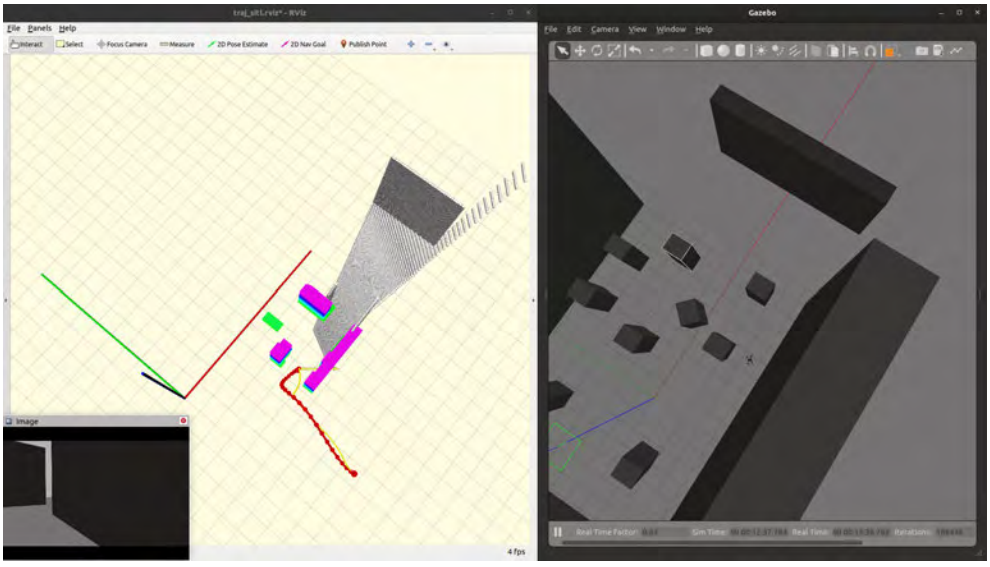


Figure 5.12: Challenges in path planning with large obstacles

Chapter 6: Conclusion

6.1 Summary of Contributions

This thesis investigates the use of UAVs in autonomous missions to advance SAR operations by developing and integrating key innovations. First, the thesis introduces a new autonomous inspection routine, specifically designed for UAVs to navigate to and inspect targets autonomously within closer proximity, identified by fiducial markers. This routine enables UAVs to perform comprehensive inspections from multiple angles without manual intervention. Second, the thesis details the integration of a custom-trained object detection model to enhance the UAV's target recognition capabilities. This model is optimized for detecting vision acuity markers, and it runs directly on the UAV's onboard computer system, providing real-time analysis during flight missions. This integration augments the UAV's operational effectiveness during the inspection of targets. Third, an extensive autonomous mission framework is created to facilitate the UAV's seamless transition between the search and inspection phases of a mission. This framework supports the UAV's autonomous exploration and precise localization of targets, utilizing the developed object detection capabilities. The implementation is validated through rigorous empirical testing across different scenarios, demonstrating the system's robustness and adaptability. Moreover, the thesis offers a comparative analysis of some of the state-of-the-art path planning algorithms, and provides an in-depth examination of one of them, critically analyzing its applica-

tion in cluttered spaces through software-in-the-loop simulation, and setting the stage for its potential real-world adaptation.

6.2 Future Work

Future work should focus on several key areas to refine and expand upon the current research:

- **Enhancement of UAV Systems**: Future work will require an upgrade of the UAV's operational capabilities by enhancing both the perception systems and on-board processing power. Integrating higher resolution cameras equipped with gimbals will significantly improve situational awareness and inspection flexibility, allowing for more dynamic adaptations in viewing angles and more precise visual coverage. Concurrently, upgrading the companion computer will address current limitations in multitasking and data processing. This will help eliminating performance bottlenecks and enabling more effective mission execution.

- **Integration of Depth Sensing in Inspection Tasks**: Depth sensing could be integrated to augment the UAV's ability to understand its spatial relationship with targets, particularly in scenarios where visual markers are partially occluded or the environment is complex. This will aid in more accurate repositioning and orientation adjustments, thereby increasing the precision and reliability of target inspections.

- **Refinement of Object Detection with Confidence Modeling**: Enhancing object detection algorithms to include temporal confidence modeling will significantly improve the robustness of detection processes. Rather than processing individual frames in isolation without

leveraging historical data from previous detections, employing advanced object tracking algorithms can facilitate the accumulation of detection confidence over time. Such work will refine the UAV's decision-making process and enable the system to accurately assess and confirm target identifications.

- **Advancement in Autonomous Exploration and 3D Mapping**: Future efforts should focus on advancing the UAV's capabilities for autonomous exploration and 3D mapping to better adapt to cluttered environments. This requires rigorous development and real-world testing of autonomous exploration algorithms coupled with the implementation of sophisticated 3D mapping technologies for better situational awareness.

Appendix A: Acuity Markers Detection Model Precision and Recall Analysis

This appendix provides further details on the model's performance evaluation through precision and recall analysis. Precision indicates the model's capability to generate relevant results over irrelevant ones, whereas recall measures the effectiveness in identifying all pertinent instances within the dataset.

Fig. A.1 shows the precision and recall curves, which further reinforce the model's reliability. To illustrate, precision portrays the model's ability to return more relevant results over irrelevant ones, while recall measures its success in finding all relevant cases in the dataset. The proximity of these curves to the top-right corner of the graph shows an optimal outcome for the model's performance by having a high level of precision and recall.

Other ways to look at the performance of the model is by looking at the Precision-Confidence curve as shown in Fig. A.2, and the Recall-Confidence curve as shown in Fig. A.3, which illustrate precision and recall levels across varying confidence thresholds. The Precision-Confidence graph shows high levels across all classes, supporting confidence in the model's predictions. Conversely, the Recall-Confidence graph shows a plateau at high recall levels to indicate the model's adeptness at identifying relevant objects without missing many. Finally, Fig. A.4 integrates precision and recall into a single metric; the F1 score. It Provides a harmonic view of the model's overall accuracy. The F1 plot suggests a balanced precision-recall relationship, which shows a

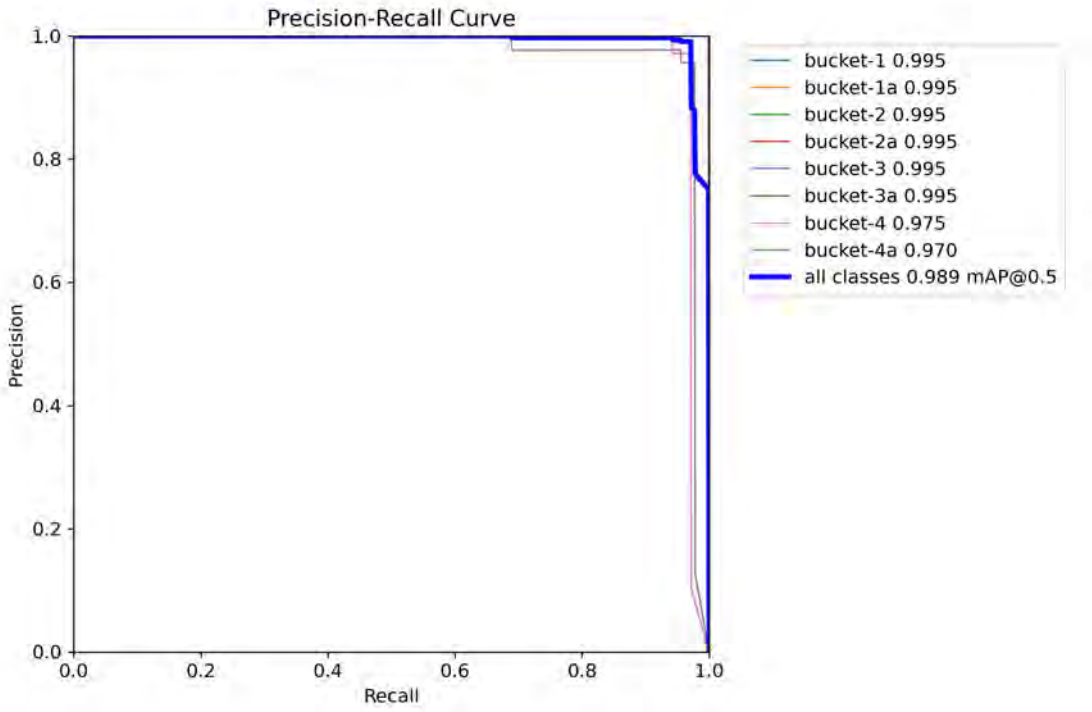


Figure A.1: Precision-Recall Curve displaying model accuracy per class

well-trained model that neither overlooks relevant objects nor outputs lots of false alarms.

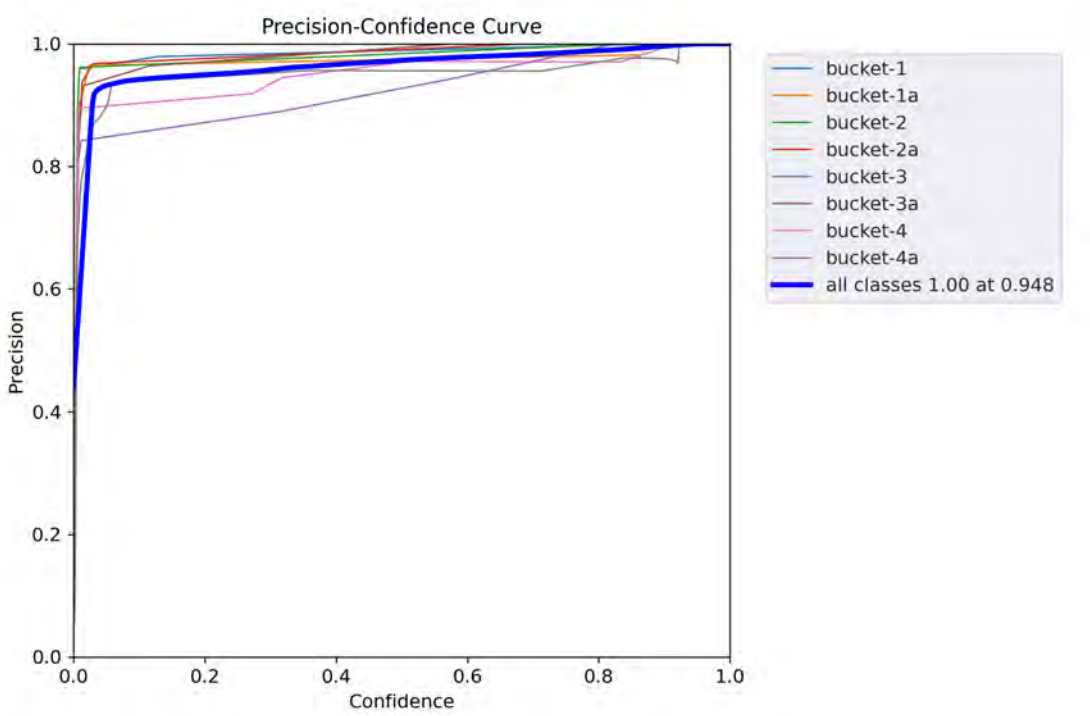


Figure A.2: Precision-Confidence Curve outlining prediction correctness vs. confidence

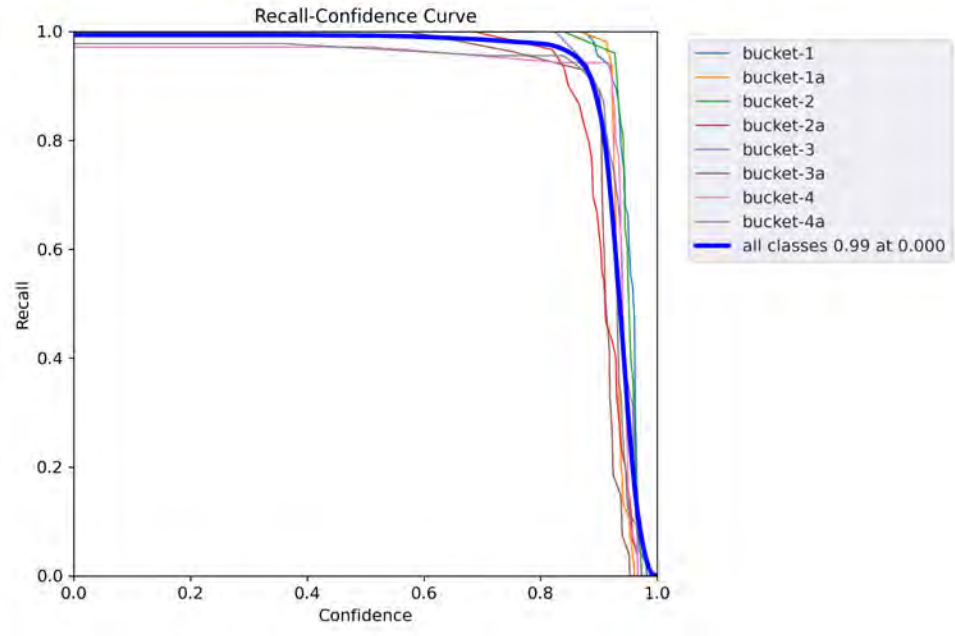


Figure A.3: Recall-Confidence Curve showing true positive detection confidence

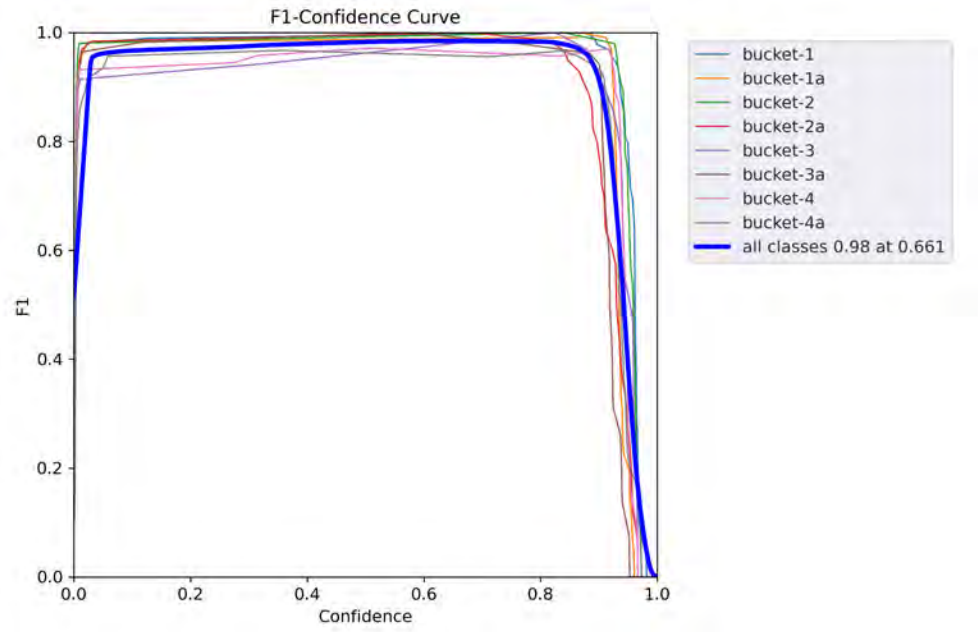


Figure A.4: F1-Confidence Curve indicating the harmonic mean of precision and recall

Bibliography

[1] Drones in the RCRC — americanredcross.github.io. `https://americanredcross.github.io/rcrc-drones/index.html`. [Accessed 11-04-2024].

[2] Konstantinos Dalamagkidis, Kimon P. Valavanis, and Les A. Piegl. On integrating unmanned aircraft systems into the national airspace system: Issues, challenges, operational restrictions, certification, and recommendations. *On Integrating Unmanned Aircraft Systems into the National Airspace System: Issues, Challenges, Operational Restrictions, Certification, and Recommendations*, pages 1–305, 1 2012.

[3] Yunus Karaca, Mustafa Cicek, Ozgur Tatli, Aynur Sahin, Sinan Pasli, Muhammed Fatih Beser, and Suleyman Turedi. The potential use of unmanned aircraft systems (drones) in mountain search and rescue operations. *The American Journal of Emergency Medicine*, 36:583–588, 4 2018.

[4] Spyridon Mavroulis, Emmanouil Andreadakis, Nafsika Ioanna Spyrou, Varvara Antoniou, Emmanouel Skourtsos, Panayotis Papadimitriou, Ioannis Kasssaras, George Kaviris, Gerasimos Akis Tselentis, Nikolaos Voulgaris, Panayotis Carydis, and Efthymios Lekkas. Uav and gis based rapid earthquake-induced building damage assessment and methodology for ems-98 isoseismal map drawing: The june 12, 2017 mw 6.3 lesvos (northeastern aegean, greece) earthquake. *International Journal of Disaster Risk Reduction*, 37:101169, 7 2019.

[5] A. Claesson, L. Svensson, P. Nordberg, M. Ringh, M. Rosenqvist, T. Djarv, J. Samuelsson, O. Hernborg, P. Dahlbom, A. Jansson, and J. Hollenberg. Drones may be used to save lives in out of hospital cardiac arrest due to drowning. *Resuscitation*, 114:152–156, 5 2017.

[6] Khaled Awad Ballous, Ahmed Nidal Khalifa, Ahmad Abdullah Abdulwadood, Mohammad Al-Shabi, and Mamdouh El Haj Assad. Medical kit: emergency drone. *https://doi.org/10.1117/12.2566115*, 11425:248–253, 4 2020.

[7] Ali Al-Naji, Asanka G. Perera, Saleem Latteef Mohammed, and Javaan Chahl. Life signs detector using a drone in disaster zones. *Remote Sensing 2019, Vol. 11, Page 2441*, 11:2441, 10 2019.

[8] Balmukund Mishra, Deepak Garg, Pratik Narang, and Vipul Mishra. Drone-surveillance for search and rescue in natural disaster. *Computer Communications*, 156:1–10, 4 2020.

[9] Mohannad Alhafnawi, Haythem A. Bany Salameh, Ala'eddin Masadeh, Haitham Al-Obiedollah, Moussa Ayyash, Reyad El-Khazali, and Hany Elgala. A survey of indoor and outdoor uav-based target tracking systems: Current status, challenges, technologies, and future directions. *IEEE Access*, 11:68324–68339, 2023.

[10] Matěj Petrlík, Tomáš Báča, Daniel Heřt, Matouš Vrba, Tomáš Krajník, and Martin Saska. A robust uav system for operations in a constrained environment. *IEEE Robotics and Automation Letters*, 5(2):2169–2176, 2020.

[11] Amr Amrallah, Ehab Mahmoud Mohamed, Gia Khanh Tran, and Kei Sakaguchi. Uav trajectory optimization in a post-disaster area using dual energy-aware bandits. *Sensors 2023, Vol. 23, Page 1402*, 23:1402, 1 2023.

[12] Ning Zhang, Francesco Nex, George Vosselman, and Norman Kerle. Training a disaster victim detection network for uav search and rescue using harmonious composite images. *Remote Sensing 2022, Vol. 14, Page 2977*, 14:2977, 6 2022.

[13] Juan Sandino, Fernando Vanegas, Felipe Gonzalez, and Frederic Maire. Autonomous uav navigation for active perception of targets in uncertain and cluttered environments. In *2020 IEEE Aerospace Conference*, pages 1–12, 2020.

[14] Juan Sandino, Fernando Vanegas, Frederic Maire, Peter Caccetta, Conrad Sanderson, and Felipe Gonzalez. Uav framework for autonomous onboard navigation and people/object detection in cluttered indoor environments. *Remote Sensing 2020, Vol. 12, Page 3386*, 12:3386, 10 2020.

[15] Carlos Sampedro, Alejandro Rodriguez-Ramos, Hriday Bavle, Adrian Carrio, Paloma de la Puente, and Pascual Campoy. A fully-autonomous aerial robot for search and rescue applications in indoor environments using learning-based techniques. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 95:601–627, 8 2019.

[16] Aditya Darshan Acharya, Subodh Bhandari, and Zekeriya Aliyazicioglu. Autonomous navigation of a quadcopter in indoor environment. *AIAA Scitech 2019 Forum*, 2019.

[17] Davide Scaramuzza and Zichao Zhang. Visual-inertial odometry of aerial robots. 6 2019.

[18] S. P.H. Driessen, N. H.J. Janssen, L. Wang, J. L. Palmer, and H. Nijmeijer. Experimentally validated extended kalman filter for uav state estimation using low-cost sensors. *IFAC-PapersOnLine*, 51:43–48, 1 2018.

[19] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3400–3407, 2011.

[20] Pengcheng Wang, Zhihong Man, Zhenwei Cao, Jinchuan Zheng, and Yong Zhao. Dynamics modelling and linear control of quadcopter. pages 498–503. IEEE, 11 2016.

[21] Patrick McNamee and Ronald M. Barrett-Gonzalez. Modeling and simulation of quadcopter dynamics in steady maneuvers. American Institute of Aeronautics and Astronautics, 1 2020.

[22] Controller Diagrams — PX4 User Guide (v1.12) — docs.px4.io. `https://docs.px4.io/main/en/flight_stack/controller_diagrams.html`. [Accessed 10-04-2024].

[23] Documentation - ROS Wiki — wiki.ros.org. `https://wiki.ros.org/`. [Accessed 09-04-2024].

[24] Simulation — PX4 User Guide (v1.12) — docs.px4.io. `https://docs.px4.io/v1.12/en/simulation/`. [Accessed 10-04-2024].

[25] Gazebo Simulation — PX4 User Guide (v1.12) — docs.px4.io. `https://docs.px4.io/v1.12/en/simulation/gazebo.html`. [Accessed 10-04-2024].

[26] Home — docs.modalai.com. `https://docs.modalai.com/`. [Accessed 10-04-2024].

[27] Chee Sheng Tan, Rosmiwati Mohd-Mokhtar, and Mohd Rizal Arshad. A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms. *IEEE Access*, 9:119310–119342, 2021.

[28] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. *Field and Service Robotics*, pages 203–209, 1998.

[29] C. Das, A. Becker, and T. Bretl. Probably approximately correct coverage for robots with uncertainty. pages 1160–1166, 12 2011.

[30] Danylo Malyuta. Guidance, navigation, control and mission logic for quadrotor full-cycle autonomy. 3 2018.

[31] Cristina Álvarez García, Sixto Cámara-Anguita, José María López-Hens, Nani Granero-Moya, María Dolores López-Franco, Inés María-Comino-Sanz, Sebastián Sanz-Martos, and Pedro Luis Pancorbo-Hidalgo. Development of the aerial remote triage system using drones in mass casualty scenarios: A survey of international experts. *PLOS ONE*, 16:e0242947, 5 2021.

[32] Hen Wei Huang, Jack Chen, Peter R. Chai, Claas Ehmke, Philipp Rupp, Farah Z. Dadabhoy, Annie Feng, Canchen Li, Akhil J. Thomas, Marco da Silva, Edward W. Boyer, and Giovanni Traverso. Mobile robotic platform for contactless vital sign monitoring. *Cyborg and Bionic Systems*, 2022, 1 2022.

[33] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[34] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[35] Glenn Jocher. Ultralytics YOLOv5. https://github.com/ultralytics/yolov5, 2020.

[36] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4:3529–3536, 7 2019.

[37] Jesus Tordesillas, Brett T. Lopez, and Jonathan P. How. Faster: Fast and safe trajectory planner for flights in unknown environments. *IEEE International Conference on Intelligent Robots and Systems*, pages 1934–1940, 3 2019.

[38] Xin Zhou, Zhepei Wang, Hongkai Ye, Chao Xu, and Fei Gao. Ego-planner: An esdf-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters*, 6:478–485, 8 2020.

[39] Xin Zhou, Jiangchao Zhu, Hongyu Zhou, Chao Xu, and Fei Gao. Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments. *Proceedings - IEEE International Conference on Robotics and Automation*, 2021-May:4101–4107, 2021.

[40] Jesus Tordesillas and Jonathan P. How. Mader: Trajectory planner in multi-agent and dynamic environments. *IEEE Transactions on Robotics*, 38:463–476, 10 2020.

[41] Boyu Zhou, Yichen Zhang, Xinyi Chen, and Shaojie Shen. Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning. *IEEE Robotics and Automation Letters*, 6:779–786, 10 2020.

[42] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *http://dx.doi.org/10.1177/0278364909359210*, 29:485–501, 1 2010.