Real-time object tracking and obstacle avoidance using UAVs

Eashwar Sathyamurthy, Nitish Ravisankar Raveendran
University of Maryland, College Park
eashwar@umd.edu, rrnitish@umd.edu

***Abstract*—This project presents a real-time object tracking and obstacle avoidance system for a VOXL-MPA-based UAV. A pre-trained YOLOv11 model detects a designated target from the onboard RGB stream, while stereo-depth fusion enables 3D obstacle localization. The offboard control loop commands the UAV to follow the tracked object and avoid nearby obstacles in real time. Object tracking is validated in both PX4 Gazebo SITL and indoor hardware tests, whereas obstacle avoidance is demonstrated through indoor hardware tests. Experimental results illustrate the feasibility of integrating real-time vision-based tracking and reactive avoidance in a lightweight onboard system.**

I. Introduction

Unmanned Aerial Vehicles (UAVs) are increasingly deployed in GPS-denied and cluttered environments, such as indoor warehouses, disaster zones, and urban canyons, where real-time perception and autonomous navigation are critical. Traditional approaches to obstacle avoidance and target tracking often depend on external motion capture systems, GPS waypoints, or pre-mapped environments, constraining their applicability in dynamic or unstructured and GPS-denied settings. [1].

Recent advances in lightweight onboard computation and deep learning-based perception enable UAVs to sense and react to their surroundings without external infrastructure. In particular, convolutional neural network (CNN) models such as YOLO (You Only Look Once) offer real-time object detection capabilities suitable for embedded systems [2]. Stereo vision systems further enhance situational awareness by estimating depth from image pairs, allowing for accurate obstacle localization [3].

This project presents an onboard prototype system that combines a pre-trained YOLOv11 model with stereo-depth fusion to enable real-time object tracking and obstacle avoidance. A VOXL-MPA drone platform is used for integration and testing, leveraging its onboard compute and depth estimation capabilities. Object tracking is validated through simulation in PX4 Gazebo and indoor flight trials, while obstacle avoidance is demonstrated in indoor hardware experiments only. The proposed pipeline advances UAV autonomy by reducing reliance on external localization infrastructure and enabling vision-based closed-loop control in real time.

II. Problem and Background

This section outlines the motivation for integrating real-time object detection and obstacle avoidance in UAV systems. While the previous section introduced the technical approach, here we focus on the practical use cases that justify this integration and the real-world challenges these capabilities address. Specifically, we highlight how real-time vision-based object detection enables UAVs to perform high-level tracking tasks, and how obstacle avoidance ensures safe navigation in dynamic, unstructured environments.

A. Real-time object detection for target tracking use cases

Real-time object detection enables UAVs to follow specific targets based on visual input, without relying on GPS tags or external beacons. This capability is essential in applications such as search and rescue, where UAVs may need to follow human subjects through debris-laden or GPS-denied environments [4]. It is also useful in inventory inspection in warehouses, where UAVs track moving objects (like AGVs or personnel) and record visual data in real time [5]. In both cases, the UAV must recognize and lock onto targets quickly and reliably despite variations in lighting, scale, and occlusion.

By leveraging models such as YOLO, which are optimized for high-speed inference with acceptable accuracy, modern UAV platforms can process RGB imagery onboard and make real-time control decisions. This moves away from the traditional paradigm of streaming data to a ground station for processing and reintroduces decision-making directly onto the vehicle.

B. Obstacle avoidance for autonomous navigation use cases

Obstacle avoidance is a foundational requirement for autonomous flight, particularly in cluttered indoor or urban environments where GPS is unreliable and obstacles are often dynamic. Applications such as autonomous warehouse patrols, drone delivery in urban settings, and infrastructure inspection demand that UAVs operate without human pilots in close proximity to physical structures [6]. In such cases, the UAV must detect, localize, and respond to obstacles in real time using onboard sensors.

Stereo vision offers a cost-effective and compact means of 3D perception, allowing the drone to estimate obstacle depth and initiate evasive maneuvers. Combined with a reactive control loop, this enables the UAV to maintain a safe trajectory while following mission goals. Unlike pre-mapped or waypoint-based navigation, reactive avoidance ensures adaptability to changing environments, making the system more robust to unforeseen conditions.

III. Technical Approach

This section details the implementation strategy for the proposed real-time object tracking and obstacle avoidance system. It is divided into three parts. First, we describe the approach for real-time obstacle avoidance using stereo vision and occupancy grid mapping. Second, we explain the object detection pipeline based on YOLOv11. Finally, we discuss how both modules are fused to enable simultaneous object following and dynamic obstacle avoidance in real time.

A. Real-time Obstacle Avoidance

1) Stereo Calibration and Depth Pipeline: To accurately estimate depth from stereo cameras, we calibrate both front and rear stereo pairs onboard using the `voxl-calibrate-camera` tool. This generates the intrinsic parameters (focal length, principal point, and distortion coefficients) and extrinsic parameters (relative rotation, translation, and baseline distance between stereo pairs). These parameters are stored in the `/data/modalai` directory and used by the `voxl-dfs-server` to rectify and align the left and right images for depth computation.

2) Depth Server and Point Cloud Topics: Once calibrated, the `voxl-dfs-server` runs in real time to compute depth maps and output a fused point cloud stream. These point clouds are bridged into ROS2 using the `voxl_mpa_to_ros2` node, which publishes the following topics:

- `/stereo_front_pc`: raw point cloud from front stereo camera (`sensor_msgs/msg/PointCloud2`)
- `/stereo_rear_pc`: raw point cloud from rear stereo camera
- `/voa_pc_out`: fused and filtered point cloud in NED frame

3) Occupancy Grid Mapping: The initial idea was to use `/voa_pc_out` for creating the occupancy grid map, but since the topic gave both front and back point clouds, separating them during drone rotation presented challenges. So, we decided to use `/stereo_front_pc` topic for creating an occupancy grid for the planner. The point clouds from `/stereo_front_pc` are processed to generate a 2D horizontal occupancy grid. We define a 5 m $\times$ 5 m area in front of the drone and discretize it into a 40×40 grid. Each cell is marked as "occupied" if one or more points fall within its boundaries. Points too close to the drone (within 0.35 m) are ignored to eliminate sensor noise and self-reflections.

We applied a median filter to filter out the noise. To ensure safety margins, we apply binary dilation to pad each occupied cell by a configurable number of grid units. This accounts for vehicle dynamics and localization uncertainty. We created a ros2 node called `occupancy_publisher` which published the topic `/occupancy_grid` with obstacles mapped onto a 40x40 grid of size 5mx5m in front of the drone along the xy (NE) axis.

4) Path Planning using Dijkstra's Algorithm: Path planning is performed on the generated occupancy grid by subscribing to `/occupancy_grid` topic using `dijkstra_planner` ros2 node, which minimizes cumulative traversal cost. Let the grid be a 2D matrix $C \in \mathbb{R}^{H \times W}$, where each cell $C_{i,j}$ is assigned:

$$C_{i,j} = \begin{cases} \infty & \text{if cell is occupied} \\ 1.0 & \text{if cell is free} \end{cases}$$

Given a start cell s and a goal cell g, we compute the minimum-cost path $\mathcal{P} = \{p_0, p_1, ..., p_n\}$ such that:

$$\mathcal{P} = \arg\min_{\mathcal{P}} \sum_{k=0}^{n-1} \text{cost}(p_k, p_{k+1})$$

where movement cost is 1.0 for axial neighbors and $\sqrt{2}$ for diagonal neighbors.

Since the planner was developed before object detection, a static goal is hardcoded into the planner for development and verification of the planner's planned path. The path is converted into NED coordinates and published as a series of trajectory setpoints using the topic `/fmu/in/trajectory_setpoint`. A separate `/planned_path` topic of type `nav_msgs/Path` is used for visualization.

The drone continually checks for obstacles at its next waypoint. If the waypoint becomes blocked due to dynamic changes in the environment, the planner replans from the current location using updated occupancy data.

This entire pipeline—from depth fusion to occupancy grid generation to Dijkstra planning—is updated at a rate of 10–30 Hz, enabling reactive avoidance while maintaining trajectory goals. The overall flow of nodes and topics used for real-time obstacle avoidance is illustrated in figure 1, showing the data pipeline from stereo input to planning output within the ROS2 framework.

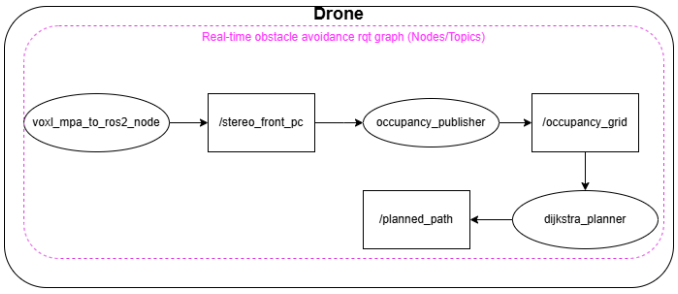


Fig. 1. Real-time obstacle avoidance `rqt_graph` showing ROS2 nodes and topic connections.

B. Real-time Object Detection

To perform object detection onboard the drone, we use the `voxl-tflite-server`, which executes a pre-trained YOLOv11 model on the RGB image stream. This model is trained on the COCO dataset and supports a wide variety of object classes. During testing, several object classes were

evaluated, but we found that the `chair` class provided the most consistent results due to its size, shape, and detection reliability under varying lighting conditions.

The object detection results are published via the VOXL middleware and bridged to ROS2 using the `voxl_mpa_to_ros2` node. Two primary topics are used:

- `/tflite`: Metadata stream about inference cycles.
- `/tflite_data`: Bounding box outputs for each detected object, published as `voxl_msgs/msg/Aidetection`.

The `Aidetection` message includes:

- `class_name`: Detected object's class (e.g., "chair").
- `x_min`, `x_max`, `y_min`, `y_max`: Bounding box coordinates in image space.
- `class_confidence`, `detection_confidence`: Confidence scores for classification and detection.

At runtime, `voxl_object_follower` ros2 node accepts the user inputs of the target object class (e.g., "chair") to follow. The system listens to `/tflite_data` and extracts the bounding box of the specified class if detected. The centroid of the bounding box is computed as:

$$x_c = \frac{x_{\min} + x_{\max}}{2}, \quad y_c = \frac{y_{\min} + y_{\max}}{2}$$

The bounding box area is used as a proxy for distance to the object. A desired area A_d is initialized from the first detection and treated as the target visual size. Future frames compute the actual area A and adjust forward motion accordingly.

Let the image dimensions be $W \times H$, and let the normalized offsets from the center be:

$$e_x = \frac{x_c - W/2}{W}, \quad e_y = \frac{y_c - H/2}{H}$$

The controller calculates the desired position offsets in the North–East–Down (NED) frame as:

$$\begin{aligned} N_d &= N + k_f \cdot \frac{A_d - A}{A_d} \\ E_d &= E + k_l \cdot e_x \\ D_d &= D + k_v \cdot e_y \end{aligned}$$

where k_f, k_l, k_v are tunable forward, lateral, and vertical gain parameters, respectively. Additionally, the yaw is adjusted to maintain the object near the image center using:

$$\psi_d = \psi + k_y \cdot e_x$$

where ψ is the current yaw and k_y is the yaw correction gain.

These computed setpoints are published as `TrajectorySetpoint` messages on `/planner/goal`. If `OFFBOARD` mode is active, the drone continuously adjusts its trajectory to maintain visual lock on the object, effectively achieving closed-loop vision-based following. The interaction between ROS2 nodes and topics for real-time object detection is shown in figure 2, highlighting the data flow from detected bounding boxes to the generation of trajectory setpoints.

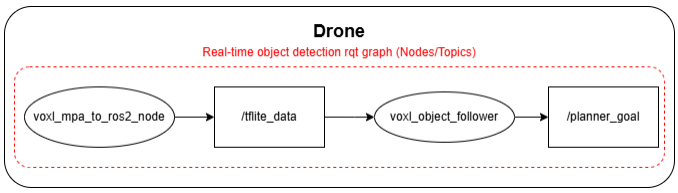


Fig. 2. Real-time object detection `rqt_graph` showing the flow from object detection messages to the object-following control loop onboard the drone.

C. Fusion of Obstacle Avoidance and Object Detection

To achieve a fully autonomous and reactive UAV system, the object detection and obstacle avoidance modules are integrated in a unified control loop. The fusion is accomplished by feeding the object tracking goal (from `/planner_goal`) into the Dijkstra-based planner, which simultaneously considers the obstacle-aware occupancy grid.

Specifically, the `voxl_object_follower` node processes bounding box detections from `/tflite_data` and publishes a dynamic goal point to `/planner/goal` based on the current position of the tracked object. This goal is updated at high frequency to maintain continuous pursuit of the target.

In parallel, the `occupancy_publisher` node creates a 2D occupancy map based on stereo vision data from `/stereo_front_pc`, filtered and padded to include safety margins. This map is published to `/occupancy_grid` and consumed by the `dijkstra_planner` node.

The `dijkstra_planner` fuses both data streams: it uses the occupancy grid to avoid obstacles and the tracking goal to guide navigation. It re-plans in real time if new obstacles are detected or the tracking goal moves, ensuring that the path remains both feasible and aligned with the target object. The resulting trajectory is published to `/fmu/in/trajectory_setpoint`, and the corresponding path is visualized via `/planned_path`.

This integrated pipeline enables the UAV to follow a moving target while dynamically avoiding obstacles in its environment—entirely onboard and without reliance on external infrastructure.

The complete flow of ROS2 nodes and topics involved in this fused architecture is illustrated in Figure 3.

IV. Results

This section presents the experimental validation of the proposed real-time object tracking and obstacle avoidance system. We divide the results into three parts. First, we demonstrate the performance of real-time obstacle avoidance, showcasing how the UAV detects and navigates around obstacles using stereo vision and a 2D occupancy grid. Second, we evaluate the object detection and tracking pipeline using onboard inference and control feedback. Finally, we present the integrated system in action, where both modules work in tandem to enable dynamic pursuit and safe navigation. The github repository can be accessed using this link and all the output videos recorded and created using rosbaging can be accessed using this google drive folder link.

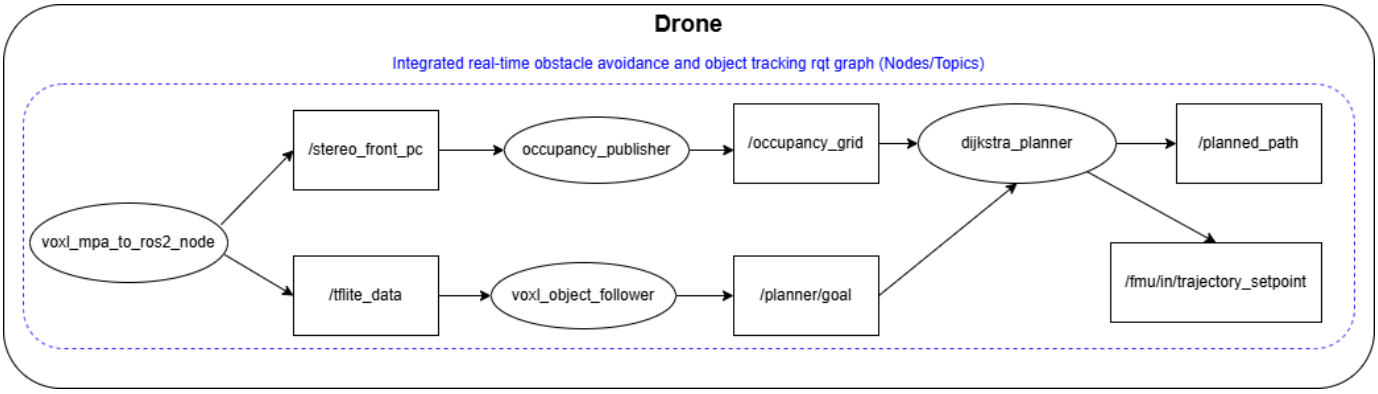


Fig. 3. Integrated real-time obstacle avoidance and object tracking `rqt_graph` showing the data flow between detection, planning, and control modules.

A. Real-time Obstacle Avoidance

To evaluate the real-time obstacle avoidance capability, the drone was flown in an indoor environment with various obstacles arranged to test navigability. Figure 5 shows the occupancy grid generated using the stereo vision input. Figure 5(a) depicts the 2D occupancy grid in RViz where black cells indicate detected obstacles. Figure 5(b) shows the same occupancy grid overlaid with the planned path generated by the Dijkstra planner, which successfully routes the drone around the obstacle cluster. Figure 5(c) provides a photograph of the actual environment setup used during indoor testing, which includes objects like chairs and cones.

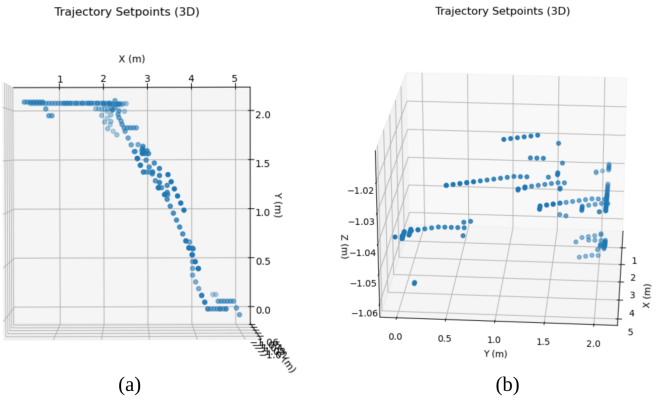
To validate the planned trajectory, we recorded and visualized the trajectory setpoints sent to the drone. Figure 4 provides both top-down and side-view perspectives of the published trajectory. Figure 4(a) shows the setpoints in the horizontal (XY) plane, confirming smooth transitions and detours around detected obstacles. Figure 4(b) offers a side view, confirming altitude consistency, which is expected since the planner operates primarily in 2D and maintains constant flight height during avoidance.

It is important to note that the trajectory setpoints represent the drone's actual executed trajectory after multiple real-time replanning events. As obstacles (and noise) are detected or previously planned waypoints become blocked, the Dijkstra planner generates new paths, resulting in deviations from the initially planned path shown in Figure 5(b). Therefore, the trajectory in Figure 4(a) may differ from the original planned path and more accurately reflects the dynamic behavior of the system in practice.

These results demonstrate that the drone can reliably perceive obstacles and generate viable paths in real-time, avoiding obstacles without external mapping or localization systems.

B. Real-time Object Tracking

To validate the object tracking capability, we tested the system in both simulation using PX4 SITL and in real-world indoor environments. In the simulation setup, the drone was placed in a Gazebo environment where a human model was introduced. The onboard YOLOv8-nano model was able to



(a) (b)

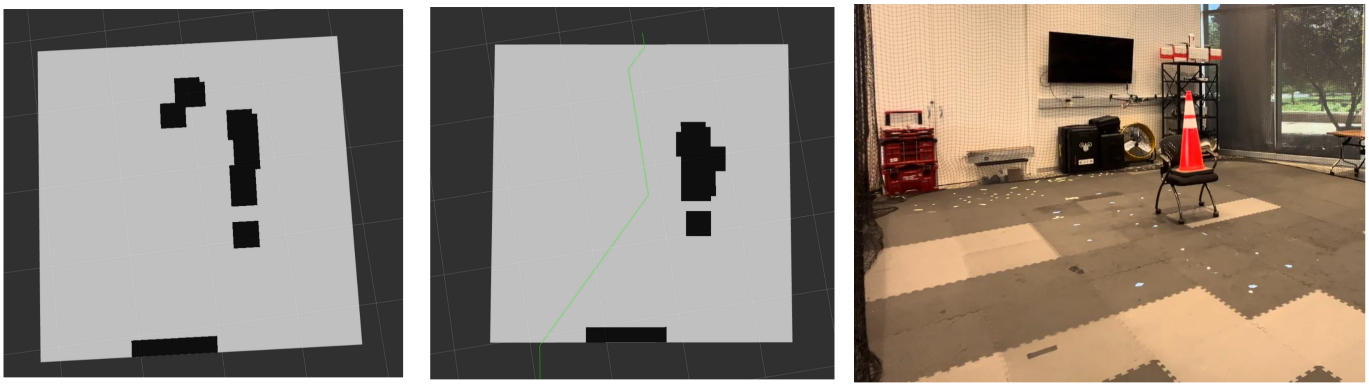
Fig. 4. Trajectory setpoints published by the planner naviating to goal (5, 0): (a) Top-down view (X-Y), (b) Side view showing constant altitude with slight variations (0.03m).

successfully detect and track the person in real time, as shown in Figure 6.

Upon selecting the target class (e.g., `person`), the drone continuously adjusted its position and yaw to maintain the object near the center of its camera frame. The controller generated trajectory setpoints that adapted to changes in the object's location and size within the image, allowing the drone to follow the target smoothly and maintain a desired standoff distance. These tests confirmed the system's ability to maintain visual lock and track moving objects under ideal simulation conditions.

To evaluate the object tracking system in real-world conditions, we conducted an indoor experiment inside the Iribe Lab. The drone was instructed to track a chair, which was moved manually across the room to simulate a dynamic target. The onboard inference and control pipeline were executed in real time, using detections from the onboard YOLOv8-nano model to generate continuous trajectory setpoints.

Figure 7 presents results from this indoor tracking trial. Figure 7(a) shows the vehicle's actual 3D trajectory (local position) as it followed the chair. Figure 7(b) visualizes the scatter plot of trajectory setpoints generated during tracking,



(a) (b) (c)

Fig. 5. Real-time obstacle avoidance results: (a) Occupancy grid from stereo front point cloud, (b) Initial Dijkstra-planned path avoiding obstacles, (c) Indoor flight setup with physical obstacles.

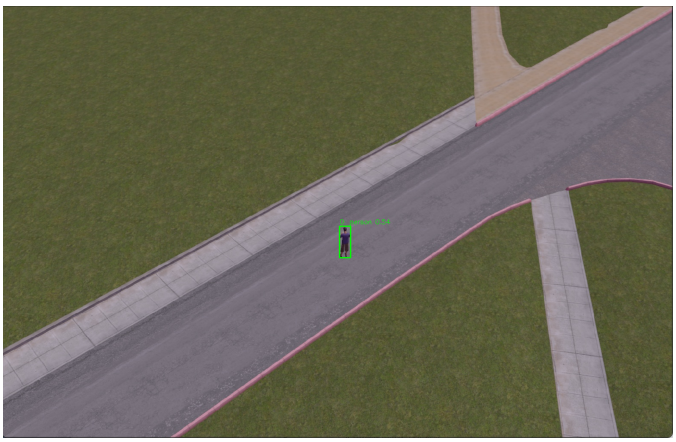


Fig. 6. Simulation-based object tracking: YOLOv8-nano detects and tracks a person in the PX4 SITL Gazebo environment.

which demonstrate continuous updates in response to the moving target. Figure 7(c) shows a live image of the chair being tracked with annotated bounding boxes, as detected by the model during flight.

These results confirm that the UAV can successfully track objects in indoor environments using onboard computation and reactive control. A video recording of this experiment will be included in the final project submission to visually demonstrate the system's performance.

C. Fusion of Obstacle Avoidance and Object Detection

To evaluate the integrated system's capability to perform simultaneous object tracking and obstacle avoidance, we designed a real-world experiment inside the Iribe Lab. Initially, a chair was used as the tracked object. However, due to the presence of multiple chairs in the environment that introduced false positives, we switched the target to a person for clearer detection.

As shown in Figure 8, the experimental setup involved placing a large obstacle (a chair with a traffic cone) in the center of the space, while a person stood on the opposite side of the room, behind the obstacle. The goal was for the drone to detect and track the person, while avoiding the obstacle in its path using onboard stereo vision and dynamic replanning.

Despite the promising integration of modules, the system encountered several practical challenges. The stereo depth camera produced noisy point clouds, especially under indoor lighting conditions, which led to inconsistencies in the occupancy grid. This was not resolved even after trying out a lot of combinations of median and uniform filters to supress the noise. Additionally, the RGB camera had limited resolution and dynamic range, causing the YOLOv11 detector to miss detections intermittently when the person moved quickly or partially occluded.

These limitations resulted in the planner frequently losing the tracking goal and attempting to replan while the obstacle remained directly in its path. This led to multiple collisions and failures in maintaining a safe trajectory. Furthermore, we observed that the onboard CPU usage was extremely high even when the drone was idle, primarily due to the load imposed by the YOLOv11 inference pipeline and point cloud processing. Figure 9 shows the system-level CPU saturation caused by multiple concurrent processes. This bottleneck limited the real-time responsiveness of the planning and control pipeline and suggests the need for more efficient computation or dedicated hardware accelerators. These outcomes highlight the sensitivity of the integrated system to perception noise and detection dropouts, and underscore the need for more robust state estimation and multi-frame detection fusion in future work.

V. Conclusion and Future Work

This project demonstrated a real-time, vision-based system for object tracking and obstacle avoidance using a VOXL-MPA UAV. We successfully integrated stereo-depth sensing, YOLO-based object detection, and reactive path planning into a unified ROS2 pipeline that ran onboard the drone without reliance on external localization infrastructure. The system was

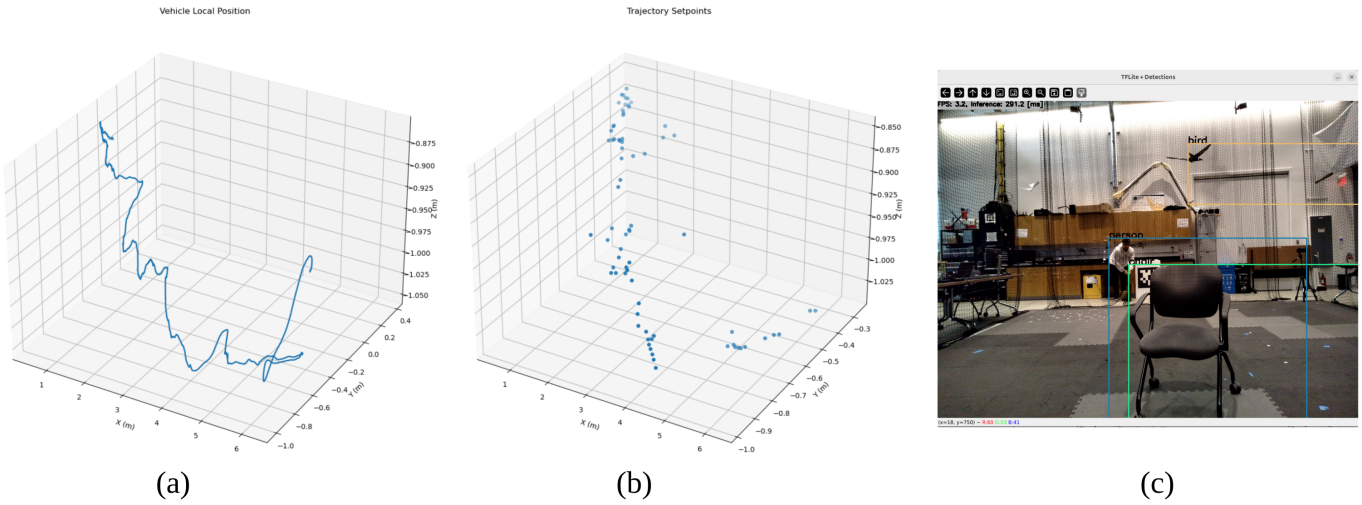


Fig. 7. Real-time object tracking in indoor environment: (a) Vehicle trajectory while tracking a chair, (b) Scatter plot of setpoints generated during tracking, (c) YOLOv11 bounding box tracking the chair in real time as the chair is moved manually.

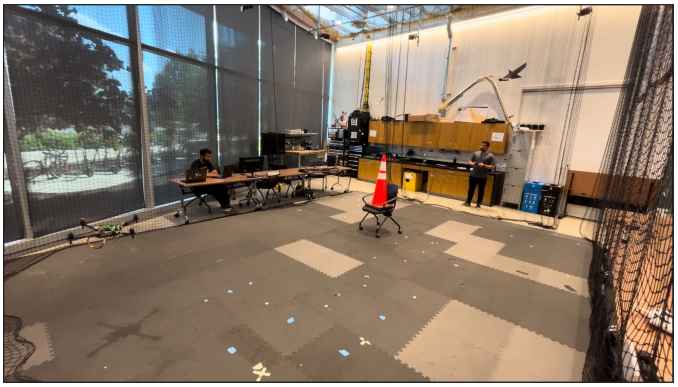


Fig. 8. Experiment setup for integrated object tracking and obstacle avoidance: The person being tracked stands behind an obstacle placed at the center of the flight space in the Iribe Lab.

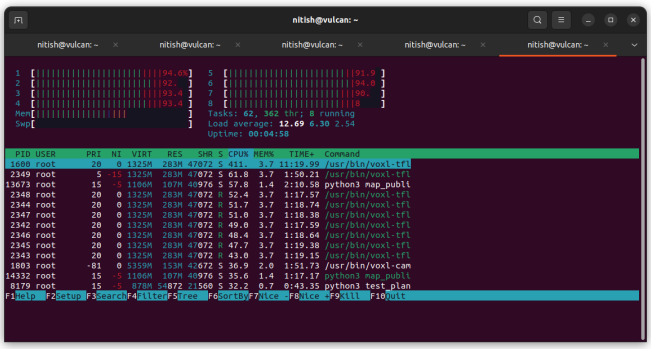


Fig. 9. CPU bottleneck observed during idle state: Multiple processes, particularly those involving `voxl_tflite_server` and and point cloud processing, consumed nearly all CPU cores even when the drone was not flying. The following results are obtained by adb'ing into the drone without the battery connected to the drone. Additional rosbaging the tflite topics would result in crashing `voxl_map_to_ros2_node`.

validated in both simulated environments (using PX4 SITL and Gazebo) and real-world indoor flight trials in the Iribe Lab.

The results showed that the object tracking module could robustly follow designated targets such as a chair or person, and that the obstacle avoidance pipeline could dynamically generate safe trajectories using local occupancy grids. We also demonstrated the feasibility of combining both capabilities in a single closed-loop control architecture that enables the drone to pursue a target while actively avoiding obstacles.

However, the fully integrated scenario revealed limitations in the current system. In particular, noisy stereo point clouds led to unstable occupancy grids, and the limited resolution of the RGB camera caused intermittent failures in detection. These issues resulted in excessive replanning and, in several cases, collisions during obstacle avoidance. The performance degradation in the presence of perception noise highlights the need for more robust estimation and control mechanisms. All the

Future Work

Several extensions are envisioned to improve the system's robustness and applicability:

- **Multi-frame detection fusion:** Integrating temporal smoothing across consecutive YOLO detections to reduce missed detections and stabilize tracking in visually ambiguous conditions.
- **Depth filtering and refinement:** Implementing noise rejection and surface normal estimation to improve the reliability of stereo-derived occupancy grids, especially in texture-poor indoor environments.
- **Persistent goal tracking:** Introducing a Kalman filter or motion model to maintain the estimated position of the target even during brief detection dropouts.
- **Integration of thermal/RGB-D cameras:** Replacing the monocular RGB feed with higher-quality RGB-D or

thermal cameras may improve both object detection and depth accuracy.

- **Real-time collision recovery:** Developing a recovery planner or fallback behavior in response to collision or detection loss to ensure continued safe operation.

REFERENCES

[1] Sturm, J., Engelhard, N., Endres, F., Burgard, W., & Cremers, D. (2012, October). A benchmark for the evaluation of RGB-D SLAM systems. In 2012 IEEE/RSJ international conference on intelligent robots and systems (pp. 573-580). IEEE.

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[3] Hirschmuller, H. (2007). Stereo processing by semiglobal matching and mutual information. IEEE Transactions on pattern analysis and machine intelligence, 30(2), 328-341.

[4] Saripalli, S., Montgomery, J. F., & Sukhatme, G. S. (2002, May). Vision-based autonomous landing of an unmanned aerial vehicle. In Proceedings 2002 IEEE international conference on robotics and automation (Cat. No. 02CH37292) (Vol. 3, pp. 2799-2804). IEEE.

[5] Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., ... & Burschka, D. (2012). Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue. IEEE robotics & automation magazine, 19(3), 46-56.

[6] Oleynikova, H., Honegger, D., & Pollefeys, M. (2015, May). Reactive avoidance using embedded stereo vision for MAV flight. In 2015 IEEE International Conference on Robotics and Automation (ICRA) (pp. 50-56). IEEE.