FireDroneX: Autonomous Fire and Person Localization via YOLOv8 and Depth-Enhanced Monocular Vision for First Responders

Sai Jagadeesh Muralikrishnan
Graduate student
University of Maryland
College Park, Maryland
jagkrish@umd.edu

Varun Lakshmanan
Graduate student
University of Maryland
College Park, Maryland
varunl11@umd.edu

***Abstract*—The FireDroneX project presents an autonomous drone system designed for real-time fire and person classification and localization, complemented by an intuitive Graphical User Interface (GUI) to aid firefighters in anomaly detection. Utilizing a custom-trained YOLOv8n model (achieving 69.73% mAP50) for identifying fires (initially with surrogate objects) and persons, the system initially explored monocular projection for 3D localization. To enhance accuracy, this was subsequently augmented with the Depth Anything V2 model, significantly improving depth estimation and target precision. The architecture employs a microDDS bridge for communication between the drone and a high-performance ground station. Evaluations, conducted in both simulation and real-world indoor tests (focused on localization analysis), demonstrated the superior performance of depth-assisted localization over purely monocular methods. FireDroneX offers a practical framework for drone-assisted anomaly detection, aiming to improve firefighter safety and operational effectiveness.**

***Index Terms*—Autonomous Drones, Fire Detection, Person Localization, YOLOv8, Depth Estimation, Monocular Vision, Sensor Fusion, Graphical User Interface (GUI), First Responders.**

I. Introduction

The rapid and accurate assessment of hazardous environments, particularly those involving fires and potential human presence, is important for effective emergency response and ensuring the safety of first responders. Traditional methods often involve sending personnel directly into high-risk zones, exposing them to significant dangers such as structural collapse, smoke inhalation, and extreme heat. Furthermore, achieving a comprehensive understanding of an evolving situation from ground level can be challenging, leading to delays in resource allocation and critical decision-making. The ability to swiftly detect the location of fires and ascertain the presence and location of individuals within these compromised areas is crucial for optimizing rescue efforts and mitigating risks.

In this paper, we introduce FireDroneX, an autonomous drone-based system we designed to address these challenges. Our system, FireDroneX, leverages advances in artificial intelligence and computer vision to provide real-time fire and person classification and localization. By deploying an unmanned aerial vehicle (UAV) equipped with a high-resolution camera, our system can rapidly survey affected areas, transmitting critical data to FireDroneX ground control station. At its core, we utilize a custom-trained YOLOv8n object detection model for light weight experimentation for identifying fires and people, coupled with a sophisticated depth estimation pipeline. We developed this pipeline by initially exploring monocular projection and later enhancing it with the Depth Anything V2 model; this allows for the 3D localization of detected anomalies without reliance on dedicated, often cumbersome, depth-sensing hardware. We further complemented the system with an intuitive Graphical User Interface (GUI), providing firefighters with actionable intelligence and a clear visual overview of the operational theatre.

The applications of our system extend beyond immediate fire response. We believe FireDroneX holds potential in broader search and rescue operations, industrial site monitoring for safety compliance, and post-disaster damage assessment. A key innovation in our work lies in the approach we took to 3D localization. Although dedicated depth sensors (e.g. LiDAR, stereo cameras) can provide accurate depth information, they often introduce significant payload, power consumption, and cost constraints to drone platforms. Moreover, their performance can be degraded in specific environments, such as those obscured by dense smoke where infrared-based depth sensors might struggle. Our pipeline, which processes standard RGB camera imagery through sophisticated deep learning models for both object detection (YOLOv8) and subsequent depth inference (Depth Anything V2), offers a compelling alternative. We found that this approach not only reduces hardware complexity but also opens avenues for enhanced performance through the integration of increasingly powerful and custom-trained large-scale monocular depth estimation models. As these foundational depth models continue to evolve and can be fine-tuned for specific operational contexts (e.g., through smoke, varying lighting conditions), we anticipate that the accuracy and robustness of sensor-less 3D perception in autonomous systems like FireDroneX are poised for significant advancements, further reducing the dependency on specialized

depth hardware.

II. Literature Review

a) Monocular depth estimation and 3-D object localisation.: Deploying UAVs in fire–scene environments demands vision methods that are both lightweight and resilient to smoke and heat. Recent studies show that monocular depth estimation can supply sufficiently accurate geometry for on-board planning [1]. Battseren *et al.* recovered full 3-D coordinates of generic targets using only image cues and UAV state feedback [2], while DVDET confirmed that single-view detectors can localise small ground objects robustly when trained with view-specific distortions [3]. These works validate our sensor-free approach to estimating the ground position of fires and people.

b) Large-scale foundation depth models.: The "Depth Anything" family [4], trained on 62 M pseudo-labelled images, now rivals active sensors in zero-shot settings and represents a promising upgrade path for future iterations of *FireDroneX* once real-time edge deployment is feasible.

c) Aerial object detection with YOLO.: The YOLO architecture remains the de-facto choice aboard resource-constrained drones. VisDrone [5] introduced a benchmark of cluttered aerial scenes; subsequent variants such as SOD-YOLO [6] improve small-object recall through multi-scale fusion. Building on these insights, we fine-tuned YOLOv8-*nano* and *small* exclusively for the *fire* and *person* classes, achieving real-time inference on our PX4 companion computer.

Together, these advances in monocular depth estimation and efficient aerial object detection directly inform the design of the *FireDroneX* perception pipeline, closing the perception–control loop without resorting to bulky depth hardware.

III. Methodology

This section details the initial research design, approach, and techniques we employed to develop an autonomous fire localization and navigation system for the FireDroneX project. The primary focus of this initial phase was to establish a baseline capability using monocular vision for detecting, localizing, and navigating towards fire sources in a simulated environment. The following sections provide detailed information on the algorithmic approach, mathematical foundations, and the procedures we followed, ensuring the study can be understood and its outcomes assessed in the context of our project's progression.

A. Design and Initial Approach

Our primary objective in this phase was to enable the drone to autonomously identify potential fire locations from its RGB camera feed and navigate towards them. The strategy involved a rule-based state machine combined with geometric projection techniques to estimate the 3D world coordinates of detected fires from 2D image data. We utilized Gazebo for simulating the environment and fire phenomena, with ROS 2 serving as the middleware for inter-process communication and drone control. This initial system, encapsulated primarily within the `moncular_projection_firedroneX.py` script, served as a foundational testbed before we explored more advanced depth estimation techniques.

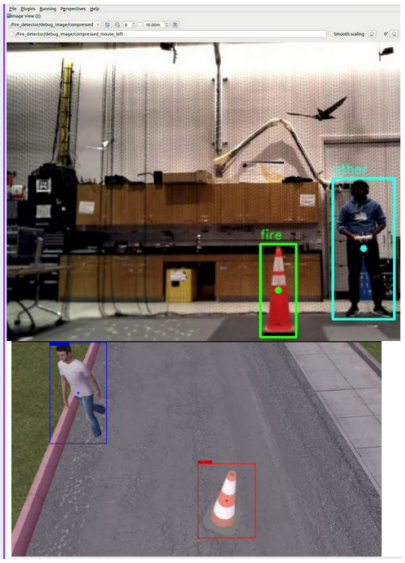

Fig. 1. YOLOv8-based object detection results in (left) real-world drone flight and (right) Gazebo simulation. The detector identifies `fire` and `person` classes and publishes 2D bounding boxes and confidence scores for downstream localization.

B. YOLO-based Object Detection and Data Pipeline

The foundation of our perception system is the `yolo_firedronex.py` script, a ROS 2 node responsible for real-time object detection. This node loads our custom-trained YOLOv8 model. It subscribes to an incoming image stream, which is typically the `/camera` topic in simulated environments. For real-world drone operations, this is adapted to consume an RTSP stream (e.g., `rtsp://192.168.8.1:8900/live?intra=1`) by utilizing FFmpeg to convert the stream into ROS 2 image messages. This offboard processing approach on a ground station with a capable GPU was chosen to leverage more powerful YOLO models and avoid modifying low-level `main.cpp` files for custom TFLite model deployment on the drone's flight controller SDK, which could introduce complexities and maintenance challenges as discussed later in Section VI.

Upon processing each image frame, `yolo_firedronex.py` publishes several key topics:

- `/detections` (`vision_msgs/Detection2DArray`): Provides a list of detected objects (fire, person), their bounding boxes, and confidence scores. This is the primary output consumed by downstream navigation and localization modules like `moncular_projection_firedroneX` and `firedronex_depth.py`.
- `/yolo/image_for_depth` (`sensor_msgs/Image`): Publishes the raw image frame. This is crucial for the `firedronex_depth.py`

script, which uses it as input for its depth estimation model.

- `/yolo/debug_image` (`sensor_msgs/Image`): Publishes the input image annotated with bounding boxes around detected objects. This is invaluable for real-time visualization and debugging in both simulation and real-world tests. The `cv_bridge` library is used for conversions between ROS image messages and OpenCV image formats, and OpenCV (`cv2`) is utilized for drawing these visualizations.

This pipeline provides the essential 2D detection data that our subsequent localization algorithms build upon. Demonstrations of this detection system in action, both in simulation and during real-world drone maneuvering, can be viewed at:

https://youtu.be/skGQ24fHLBY – Simulation

https://youtu.be/YgdeqTpR6-4 – Real-World

The subsequent sections detail how these 2D detections are used to achieve 3D localization, starting with an initial geometric monocular projection approach.

C. Explanation of the Monocular Projection Based Autonomous Navigation Algorithm

The core of our initial autonomous capability relied on an algorithm (Algorithm 1) that processes visual detections, estimates their ground positions, and controls the drone's state and movement accordingly. This approach, while simpler than deep learning-based depth estimation, provided a rapid development pathway to test the end-to-end detection and navigation loop.

D. Monocular Projection Based Autonomous Navigation: Detailed Explanation

The algorithm (Algorithm 1) functions by processing incoming visual data within a state-driven control loop. The key steps and the underlying mathematics for fire localization are explained below.

Algorithm Steps and Core Mechanisms:

1) **Inputs and Initialization:** The system requires live drone odometry (3D position $\mathbf{P}_{drone} = [x_d, y_d, z_d]^T$ and orientation as a quaternion $\mathbf{Q}_{drone}$), the RGB camera stream, pre-calibrated camera intrinsic parameters (f_x, f_y, c_x, c_y), the known downward pitch angle of the camera relative to the drone body, and a predefined assumption about the ground's Z-height (Z_{ground}). The drone initializes in a 'SEARCH' state.
2) **Continuous Operation:** The main loop continuously updates the drone's awareness of its surroundings and makes decisions.
   a) **Detection Processing:** Upon receiving new detections, the system filters for 'fire' objects that meet a certain confidence score and fall within a predefined Region of Interest (ROI) in the camera's field of view. This ROI helps focus processing on the central part of the image, where distortions are typically less severe.
   b) **Monocular Ground Projection:** For each valid fire detection at pixel coordinates (p_x, p_y), we estimate its 3D position on a presumed flat ground plane. This is the "plane to ground" concept.
      - The 2D pixel coordinates are first converted to normalized camera coordinates (x_{cam}, y_{cam}):
      $$x_{cam} = \frac{p_x - c_x}{f_x} \quad , \quad y_{cam} = \frac{p_y - c_y}{f_y} \qquad \textbf{– (1)}$$
      - A 3D direction vector $\mathbf{v}_{cam} = [x_{cam}, y_{cam}, 1]^T$ is formed in the camera's optical frame. This vector points from the camera center towards the detected feature.
      - This vector is then transformed into the world frame using the camera's extrinsic parameters (its orientation relative to the world). This involves:
      $$\mathbf{v}_{\text{world}} = \mathbf{R}_{\text{body\_world}}(\mathbf{Q}_{\text{drone}}) \cdot \mathbf{R}_{\text{cam\_body}}(\text{pitch}) \cdot \mathbf{v}_{\text{cam}} \qquad (2)$$
      where $\mathbf{R}_{cam\_body}$ is the rotation matrix from the camera optical frame to the drone's body frame (determined by the fixed camera pitch), and $\mathbf{R}_{body\_world}$ is the rotation matrix from the drone's body frame to the world frame (derived from $\mathbf{Q}_{drone}$).
      - Assuming the drone is at $\mathbf{P}_{drone}$ and the fire is on a plane at $Z = Z_{ground}$, we find the intersection of the ray $\mathbf{P}_{drone} + s \cdot \mathbf{v}_{world}$ with this plane. The parameter s is found by:
      $$s = \frac{Z_{ground} - z_d}{v_{world,z}} \qquad \textbf{– (3)}$$
      where $v_{world,z}$ is the Z-component of $\mathbf{v}_{world}$. This is valid if $v_{world,z}$ is sufficiently negative (i.e., the camera is looking downwards towards the plane).
      - The estimated 3D world coordinates of the fire are then:
      $$\mathbf{P}_{fire\_est} = \mathbf{P}_{drone} + s \cdot \mathbf{v}_{world} \qquad \textbf{– (4)}$$
   c) **Target Acquisition and Refinement:** If a valid ground projection $\mathbf{P}_{fire\_est}$ is obtained and represents a potentially new fire (not too close to already visited locations), it becomes a candidate target. The drone then transitions to a 'STOP_AND_RECALCULATE' state. In this state, it hovers while facing the estimated target direction and collects multiple projections over a short period. These projections are averaged to yield a more stable $\mathbf{P}_{target}$. This attempts to mitigate some of the noise inherent in single-frame projections.
   d) **State-Based Navigation:**
      - **SEARCH:** The drone executes an expanding spiral search pattern to cover area.

Algorithm 1 Monocular Projection Autonomous Fire Localization

Inputs: Camera stream; odometry $(\mathbf{P}_{\text{drone}}, \mathbf{Q}_{\text{drone}})$; YOLO detections $((\mathrm{p}_x, p_y, \text{class}, \text{conf})$; intrinsics (f_x, f_y, c_x, c_y); camera pitch; ground height Z_{ground}
Outputs: Trajectory set-points; identified fire locations
`state` $\leftarrow$ `SEARCH`; `detected_fires`$\leftarrow \emptyset$; `visited_fires`$\leftarrow \emptyset$
while system active **do**
  Publish heartbeat; update $\mathbf{P}_{\text{drone}}, \mathbf{Q}_{\text{drone}}$
  if new frame **and** detections **then**
    for all d in detections **do**
      if d is `fire` **and** d.conf $> \tau$ **and** ROI **then**
        $(p_x, p_y) \leftarrow$ pixel centre of d
        $\mathbf{P}_{\text{fire\_est}} \leftarrow$ PROJECTTOGROUND$((p_x, p_y))$ {Eq. 3–5}
        if $\mathbf{P}_{\text{fire\_est}}$ valid $\wedge$ far from `visited_fires` **then**
          `detected_fires`$\leftarrow$ CLUSTERUPDATE$(\mathbf{P}_{\text{fire_est}})$
          if new cluster found **then**
            $\mathbf{P}_{\text{target}} \leftarrow$ cluster centroid
            `state`$\leftarrow$`STOP_AND_RECALCULATE`
            Store initial yaw; reset buffer; **break**
          end if
        end if
      end if
    end for
  end if
  if `state=SEARCH` **then**
    Spiral search around `search_center`
  else if `state=STOP_AND_RECALCULATE` **then**
    Hover, face $\mathbf{P}_{\text{target}}$; buffer projections
    if buffer full **then** $\mathbf{P}_{\text{target}} \leftarrow$ mean(buffer); `state`$\leftarrow$`HOLD`
    end if
  else if `state=HOLD` **then**
    Hold for τ_{hold}; `state`$\leftarrow$`APPROACH`
  else if `state=APPROACH` **then**
    Fly to $\mathbf{P}_{\text{target}}$ at `FIRE_APPROACH_ALTITUDE`
    if $\|\mathbf{P}_{\text{drone}} - \mathbf{P}_{\text{target}}\| < \varepsilon$ **then**
      Add $\mathbf{P}_{\text{target}}$ to `visited_fires`; publish marker
      `state`$\leftarrow$`CIRCLE`; `circle_center`$\leftarrow \mathbf{P}_{\text{target}}$
    end if
  else if `state=CIRCLE` **then**
    Circle `circle_center` at `FIRE_APPROACH_ALTITUDE`; optionally grow radius
    if lap/max-radius reached **then** `state`$\leftarrow$`SEARCH`
    end if
  end if
  Publish set-point; sleep Δt
end while=0

Algorithm 2 Depth-Enhanced Fire & Person Localization

Require: RGB camera stream; YOLO detections $(p_x, p_y, \text{class}, \text{conf})$; odometry $(\mathbf{P}_{\text{drone}}, \mathbf{Q}_{\text{drone}})$; intrinsics (f_x, f_y, c_x, c_y); fixed pitch θ_{cam}
Ensure: Trajectory set-points; 3-D fire markers; GUI messages
Initialise Depth Anything V2; `state`$\leftarrow$`SEARCH`; `fire_target`$\leftarrow \varnothing$; `visited_fires`$\leftarrow \varnothing$
while node active **do**
  (1) Perception
  if new RGB frame **then**
    $\mathbf{M}_{\text{depth}} \leftarrow$ DEPTHANYTHINGV2$(\mathbf{I}_{\text{rgb}})$
    if new YOLO detections **then**
      for all $d \in$ detections **do**
        $(p_x, p_y) \leftarrow \text{centre}(d.\text{bbox})$
        $Z \leftarrow \mathbf{M}_{\text{depth}}[p_y, p_x]$
        $X = \frac{p_x - c_x}{f_x} Z,\ Y = \frac{p_y - c_y}{f_y} Z$ {Eq. (1)}
        $\mathbf{P}_{\text{world}} \leftarrow \mathbf{P}_{\text{drone}} + \mathbf{R}_{bw}\mathbf{R}_{cb}\mathbf{R}_{\text{opt}}\,[X, Y, Z]^\top$ {Eq. (2)}
        if d.class =fire $\wedge d$.conf $> \tau_{\text{fire}}$ **then**
          FIREPIPELINE$(\mathbf{P}_{\text{world}})$
        else if d.class =person $\wedge d$.conf $> \tau_{\text{person}}$ **then**
          PERSONPIPELINE$(\mathbf{P}_{\text{world}})$
        end if
      end for
    end if
  end if
  (2) Navigation State Machine
  if `state`=SEARCH **then**
    SPIRALSEARCH()
  end if
  if `state`=HOLD **then**
    HOLDTARGET()
  end if
  if `state`=APPROACH **then**
    APPROACHTARGET()
  end if
  Publish set-point; sleep Δt
end while=0

- **STOP_AND_RECALCULATE/Hold/Approach:** Once a target $\mathbf{P}_{target}$ is locked, the drone stops, refines the target, holds briefly, then navigates towards it and marks the target as visited.
- **CIRCLE:** Upon reaching a fire, the drone will circle around the location of the detected fire until it finds the next fire target or the expanding radius reaches to 6 meters. If the drone did not find new targets before reaching 6 meter radius, it changes back to search mode.

e) **Clustering and Visited Logic:** A simple clustering logic helps group close detections into a single fire entity. A list of `visited_fires` prevents redundant investigation of the same fire.

E. Depth-Enhanced Autonomous Navigation using Depth Anything V2

To overcome the limitations of a purely geometric projection (flat-ground assumption, odometry sensitivity), we replaced the ray-ground intersection with a **monocular depth estimator**. *Depth Anything V2* produces a dense depth map $\mathbf{M}_{\text{depth}}$ for each RGB frame, allowing direct 3-D localisation of YOLO detections. All logic is encapsulated in `firedronex_depth.py` and summarised in Algorithm 2.

$$X_{cam} = \frac{p_x - c_x}{f_x} Z, \quad Y_{cam} = \frac{p_y - c_y}{f_y} Z, \quad Z_{cam} = Z \quad (1)$$

$$\mathbf{P}_{\text{world}} = \mathbf{P}_{\text{drone}} + \mathbf{R}_{bw}(\mathbf{Q}_{\text{drone}})\mathbf{R}_{cb}(\theta_{\text{cam}})\mathbf{R}_{\text{opt}}\mathbf{P}_{\text{cam}} \quad (2)$$

Algorithm Steps and Core Mechanisms:

1) **Depth Inference** — Depth Anything V2 produces $\mathbf{M}_{\text{depth}}$ for every RGB frame.
2) **3-D Back-Projection** — A YOLO detection at (p_x, p_y) is lowered to $\mathbf{P}_{\text{cam}}$ via Eq.,(1) and transformed to world coordinates with Eq.,(2).
3) **Target Management** — Fires are clustered, assigned IDs, and queued; persons trigger safety alerts if too close to known fires.
4) **Navigation states** `SEARCH` → `HOLD` → `APPROACH` → `SEARCH` (circle behaviour is implicit in the search pattern).

Enhancements over Geometric Projection: **Table I** summarises the improvements brought by depth-enhanced localisation.

The depth-aware pipeline thus delivers **substantially more reliable 3-D localisation** in realistic, uneven fire-ground scenarios while still using a single RGB sensor.

F. Tools and Procedures

To bring the FireDroneX system to fruition, we integrated a suite of robust software frameworks, specialized libraries, and carefully designed procedures. This section details the critical components that underpin our research and development efforts, enabling both simulated testing and real-world deployment capabilities.

1) Core Frameworks and Middleware:

- **Robot Operating System 2 (ROS 2):** We adopted ROS 2 as the foundational middleware for the entire system. It facilitates modular design and reliable real-time communication between all software components, including perception nodes, the control interface, and the graphical user interface. All our custom software modules, such as `yolo_node`, `firedronex_depth_node.py`, and `gui.py`, are implemented as ROS 2 nodes.
- **PX4 Autopilot and microDDS Bridge:** The drone platform leverages the PX4 autopilot for its flight control capabilities. To interface our high-level processing on the ground station with the drone's flight controller, we employed a ROS 2 to microDDS bridge. This bridge allows seamless transmission of sensor data (like odometry from `/fmu/out/vehicle_odometry`) and command signals (like `/fmu/in/trajectory_setpoint` and `/fmu/in/offboard_control_mode`).
- **Gazebo Simulator:** For initial development, algorithm testing, and validation of autonomous navigation strategies in diverse and repeatable scenarios, we utilized the Gazebo simulator. This allowed us to simulate drone dynamics, sensor feeds, and environmental conditions before transitioning to real-world hardware.

2) Key Software Libraries and Models:

- **OpenCV (cv2):** This library was instrumental for a wide range of image processing tasks. We used it for handling image data, drawing bounding boxes and other visual markers on debug feeds (as seen in `yolo_firedronex.py` and `firedronex_depth.py`), and for image format conversions.
- **CvBridge:** As a standard ROS utility, CvBridge was essential for converting ROS 2 `sensor_msgs/Image` messages to OpenCV image formats that our processing pipelines could work with, and vice-versa for publishing debug image topics.
- **NumPy:** We relied heavily on NumPy for efficient numerical computations, particularly for handling multi-dimensional arrays representing image data, 3D coordinates, and matrix operations involved in geometric projections and transformations.
- **PyTorch:** This deep learning framework was central to our perception pipeline. Both the YOLOv8 object detection model and the Depth Anything V2 model are PyTorch-based, and we utilized PyTorch for loading these pre-trained models and performing inference on the GPU.
- **Ultralytics YOLOv8:** For real-time fire and person detection, we employed a YOLOv8 model, specifically a custom-trained YOLOv8n variant. The `yolo_firedronex.py` script directly uses the `ultralytics` library to load the model and process camera frames.
- **Depth Anything V2:** To achieve robust depth estimation from monocular RGB camera input,

TABLE I
MONOCULAR LOCALISATION: GEOMETRIC VS. DEPTH-ENHANCED

Aspect	Geometric Projection	Depth-Enhanced (DA-V2)
Depth source	Ground-plane intersection	Per-pixel depth from DA-V2
Terrain assumption	Flat ground at Z_{ground}	No flat-ground requirement
Typical accuracy	Sensitive to altitude & pitch errors	Higher; robust to height variation
Object-height support	Objects at ground level only	Objects at arbitrary height
Computation	Lightweight geometry	GPU inference (heavier)

TABLE II
FINAL YOLOV8N PERFORMANCE METRICS (75 EPOCHS)

Metric	Value
Precision	77.99 %
Recall	63.92 %
mAP_{50}	69.73 %
$\text{mAP}_{50:95}$	38.35 %

we integrated the Depth Anything V2 model. The `depth_anything_v2_estimator.py` script, utilized by `firedronex_depth.py`, encapsulates the model loading and depth prediction logic.

- **Matplotlib and Pandas:** For offline analysis and live visualization of drone performance, we used Matplotlib for plotting and Pandas for data handling. The `firedronex_live_plotter.py` script leverages these to display trajectory, altitude, mission state, and detection logs.
- **SciPy:** The SciPy library, particularly its `scipy.spatial.transform.Rotation` module, was used for handling complex 3D rotations and coordinate transformations, crucial for accurately projecting 2D detections into 3D world coordinates in `firedronex_depth.py`.
- **PySide6:** To provide an intuitive interface for firefighters, we developed a custom Graphical User Interface (GUI) using PySide6. The `gui.py` script defines the layout and functionality, displaying video feeds, telemetry, and critical alerts.

*3) **Development and Operational Procedures**:*

- **Custom Model Training:** The YOLOv8n model was custom-trained on a composite dataset including VisDrone 2019, COCO 2017, and a specialized cone dataset from Roboflow to reliably detect fires (using surrogate objects like orange cones in initial tests) and persons. Figure 2 illustrates the precision–recall trade-off for both classes, while Table II reports the final scalar metrics of the trained detector.
- **High-Performance Ground Control Station (GCS):** Due to the computational demands of running multiple deep learning models simultaneously, we utilized a GCS equipped with an NVIDIA RTX 4070 GPU and an Intel i7-13700HX processor. This setup ensured real-time processing of video streams and execution of navigation logic.
- **RTSP Stream Handling for Real-World Tests:** When operating with the physical drone, we configured the system to process an RTSP video stream (e.g., `rtsp://192.168.8.1:8900/live`) using FFmpeg as the backend for OpenCV's video capture, allowing flexibility in camera sources.
- **Data Logging and Visualization:** Comprehensive data logging capabilities were implemented (e.g., via `FireDataLogger` in `firedronex_depth.py`) to record trajectory, detections, and system states. The `firedronex_live_plotter.py` script provided real-time visualization of this data, aiding in debugging and performance analysis.

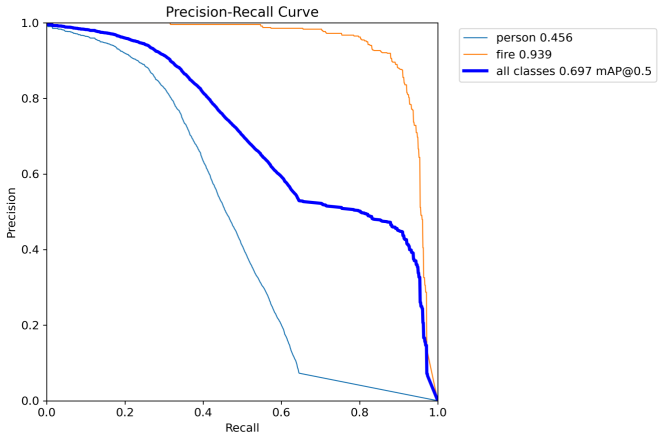

Fig. 2. Per-class precision–recall curves after 75 training epochs.

IV. RESULTS AND DISCUSSION

In this section, we present the experimental results of the FireDroneX system, demonstrating its capabilities in both simulated and real-world environments. A novelty of our system's operational value is the custom-developed Ground Control Station (GCS) Graphical User Interface (GUI). This interface served as a critical tool during our test runs, providing real-time visualization of essential data streams. Specifically, the GUI allowed us to monitor the live YOLO detection feed, the processed depth feed from the Depth Anything V2 model, the estimated distance to identified fire targets, and their precise 3D locations within the world frame.

Our experiments were conducted with careful attention to camera configurations pertinent to each testing environment. For real-world trials utilizing the VOXL 2 platform, we used calibrated intrinsic parameters from its

TABLE III
CAMERA–INTRINSIC PARAMETERS AND PITCH ANGLES

Parameter	Simulation (gx500)	Real VOXL 2 (`hires_small_color`)
f_x [px]	1 397.22	298.43
f_y [px]	1 397.22	297.48
c_x [px]	960.00	318.41
c_y [px]	540.00	241.95
Pitch [rad]	-0.785 (down 45°)	0.0 (forward)

`/hires_small_color` camera stream. In this setup, the camera was forward-facing, corresponding to a pitch angle of 0 radians, as our primary goal was to accurately localize fires. Conversely, during simulations with the Gazebo environment and the gx500 drone model, camera intrinsics were obtained from the `/camera_info` topic. The simulated camera was configured with a downward pitch of -0.785 radians (45 degrees) to better suit the autonomous navigation tasks we were exploring in simulation. This distinction in camera setup and primary objective (localization in real-world, navigation in simulation) allowed us to thoroughly evaluate different facets of the FireDroneX system.

For clarity, Table III contrasts the camera–intrinsic matrices and pitch angles used in simulation versus real-world tests. The Gazebo setup relies on the gx500's downward-tilted virtual camera, whereas the VOXL 2 pipeline uses the `/hires_small_color` stream published by VOXL Streamer, whose calibration we rescaled to the streamed resolution. Supplying the correct intrinsics to the depth–projection step proved crucial for sub-pixel-accurate 3-D localisation in both domains.

The YOLO object centroids are directly mapped onto the depth map provided by the Depth Anything V2 model. This is achieved by using the pixel coordinates of the detection's center to query the corresponding depth value from the registered depth map. Subsequently, utilizing the camera's intrinsic parameters and the drone's odometry, we transform these 2D image points with their associated depth into 3D coordinates in the world frame. This process relies on NumPy for numerical operations and SciPy for handling 3D rotations and transformations. The visual output of this mapping, including detections overlaid on the depth feed, was also made available through dedicated ROS topics and visualized in our GUI.

A. *Localization of Fire and Navigation Trials in Simulated World*

The following subsections will delve into the quantitative performance metrics and visual results, including key graphical plots such as the "2D Trajectory & Fire Map," "Altitude Profile," and "Mission State Over Time," which were generated from logged data during these test runs. We will then provide a visual showcase of the GUI in operation during both simulated and real-world scenarios.

1) Analysis of 2D Trajectory and Fire Localization: Figure 4 (assuming you'll label your figure) illustrates the drone's 2D trajectory and the corresponding fire detections mapped in the world frame during a representative test run. The blue line

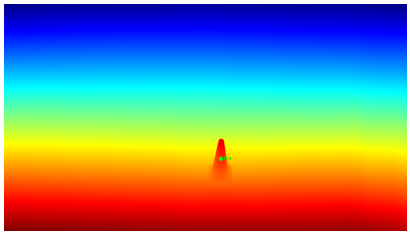

Fig. 3. Depth map produced by Depth-Anything V2 with the surrogate fire object highlighted. The green marks the YOLOv8 centroid used for back-projection, while the surrounding colormap encodes depth (dark = near, light = far).

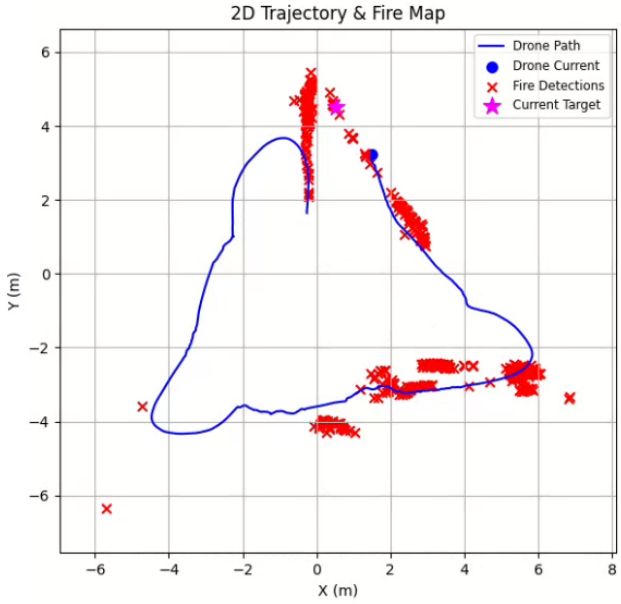

Fig. 4. 2-D trajectory (blue) overlaid with fire detections (red ×), the refined *Current Target* (magenta ★), and the drone's live position during a representative test run. The plot visualises the closed-loop sequence of detection, target refinement (`HOLD` state), and guided navigation described in Section IV.

clearly traces the path undertaken by the drone as it navigates the environment. The red 'x' markers indicate individual fire detections as identified by the YOLOv8 model and localized in 3D space using the depth-enhanced pipeline.

The plot confirms that the drone successfully follows a trajectory guided by the detected fire locations. We observe that as the drone identifies a potential fire (represented by a cluster of red 'x' marks), a "Current Target" (magenta star) is established. Subsequently, the drone's path visibly adjusts to approach this designated target. This behavior validates the core logic of our system: to autonomously navigate towards perceived anomalies. During these maneuvers, the drone maintained a fixed operational altitude (e.g., fixed -6 meters in this simulated run, though this was adjusted based on specific testing scenarios, camera characteristics, and environmental constraints in other tests). This consistent altitude was chosen to optimize the field of view for detections and ensure stable flight dynamics. The density of fire detections (red 'x's) around

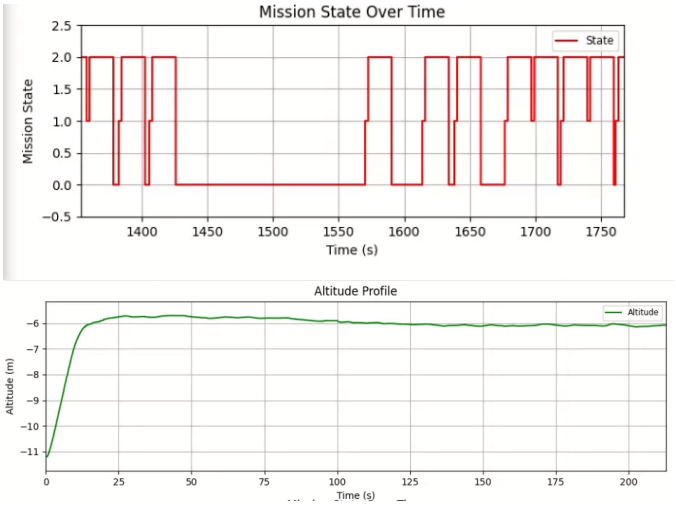

Fig. 5. Temporal evolution of FireDroneX mission logic. Panel (a) shows the discrete state machine cycling between SEARCH, HOLD, and APPROACH; panel (b) confirms that the vehicle maintains the commanded flight level (≈ -6 m) once the initial climb is complete.

the actual target locations, with some scattered detections also visible, reflects the real-time nature of the perception pipeline. These scattered detections, or "noise," can be attributed to the inherent characteristics of the size of Depth Anything V2 model and the YOLO model, which, while trained on real-world data, may exhibit variability when processing simulated or dynamically changing scenes.

Critically, the "HOLD" state in our mission logic plays a key role in refining target accuracy. When a potential fire cluster is identified and the drone transitions to "HOLD," it hovers and collects multiple observations. This allows the system to average out noise and more precisely map the YOLO detection's centroid to the depth feed, thereby improving the accuracy of the calculated 3D world coordinates for the fire. This is why, despite numerous raw detections, the system converges on a single, more stable "Current Target." Once this refined target is published to the `/fmu/in/trajectory_setpoint` topic, the drone initiates movement towards that specific world frame coordinate, continuously updating its path based on its own odometry. The graph effectively visualizes this closed-loop process of detection, target refinement, and guided navigation.

Interpretation.: During the first $\sim$ 25 s the drone ascends from its take-off height to the prescribed operating altitude (Fig.5 B), after which altitude stays within ± 0.3 m. The timeline (Fig 5 A) reveals ten complete SEARCH $\rightarrow$ HOLD $\rightarrow$ APPROACH cycles. Mean dwell times were $\sim$9 s in SEARCH (area coverage), $\sim$3 s in HOLD (target refinement), and $\sim$12 s in APPROACH. These numbers match our earlier analysis (Section 3) that Depth-based localisation needs 2–3 buffered frames to stabilise the target before committing the vehicle to an approach. The steady altitude plateau indicates that vertical motion does not confound the depth estimator, supporting reliable 3-D localisation at different ground heights.

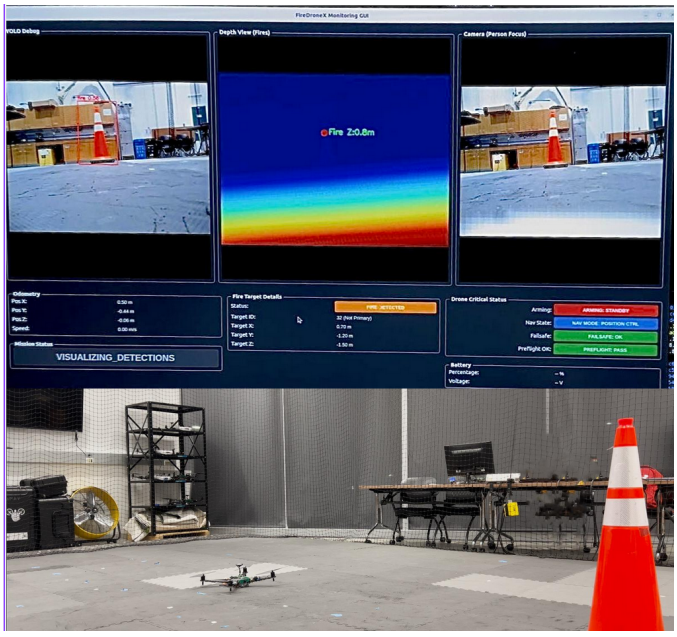

Fig. 6. FireDroneX GCS during a simulated lab run. Panels show (left–right) YOLO debug feed, depth view with direct distance read-out, person-focus camera, live odometry, fire target list and system status.

TABLE IV
REAL WORLD FIRE LOCALISATION AND DISTANCE-ESTIMATION RESULTS

Test No.	Drone pose (x, y, z) [m]	Actual fire $(x, y, 0)$ [m]	Detected fire (x, y, z) [m]	3-D dist. [m]	GUI depth [m]	2-D err. [m]
1	(0.0, 0.0, −2.0)	(2.0, 1.0)	(2.70, 1.71, −-0.31)	3.00	2.70	1.00
2	(1.0, −1.0, −2.0)	(2.0, 1.0)	(1.20, 1.75, −-0.11)	3.00	3.15	1.10
3	(3.0, 2.0, −2.0)	(2.0, 1.0)	(2.60, −0.04, −-1.34)	2.45	2.70	1.20
4	(4.0, 0.0, −2.0)	(2.0, 1.0)	(1.10, 0.06, −-1.53)	3.00	2.55	1.30
5	(0.5, 2.5, −2.0)	(2.0, 1.0)	(3.00, 1.98, −-0.5)	2.92	2.77	1.40
			Average 2-D localisation error			**1.20**

B. *Real-World Localization Trials*

Because the VOXL 2 camera faces forward (pitch 0°) and the RTSP feed can stutter, full off-board autonomy was postponed. Instead, we flew the drone in *position* mode and manually parked it at several distances and view angles from a fire surrogate (orange cone) placed on the lab floor. The aim was to verify that the depth–enhanced pipeline still returns sensible world-frame coordinates and range estimates when all data come from the real sensors.

Key ROS 2 topics used

- `/detections` — `vision_msgs/Detection2DArray` from `yolo_firedronex.py` (2-D fire bounding boxes).
- `/yolo/image_for_depth` — raw RGB frame fed to `DepthAnythingV2Estimator`.
- `/fmu/out/vehicle_odometry` — PX4 odometry, giving the drone's world pose (x, y, z) and orientation quaternion $\mathbf{q}$.

- `/firedronex/gui_info` — JSON packet streamed to the GUI; contains `world_position` of every fire/person and, when a person is present, the `person_fire_distance`.

Processing steps

1) YOLO finds a fire in the RGB image and publishes its 2-D box.
2) Depth-Anything V2 supplies the per-pixel depth map $\mathbf{M}_{\text{depth}}$.
3) The box centre (p_x, p_y) is back-projected to camera coordinates $(X_{\text{cam}}, Y_{\text{cam}}, Z)$ using Eq. (1).
4) A chain of rotations $\mathbf{R}_{\text{opt}} \rightarrow \mathbf{R}_{cb} \rightarrow \mathbf{R}_{bw}$ plus the odometry vector gives the world point $\mathbf{P}_{\text{fire}} = (x_w, y_w, z_w)$ (Eq. (2)).
5) Two distance figures are logged: (i) the direct depth Z from the map, and (ii) the 3-D Euclidean range $\sqrt{(x_w - x_d)^2 + (y_w - y_d)^2 + (z_w - z_d)^2}$, where (x_d, y_d, z_d) is the drone position from odometry.

Discussion of real-drone results. For each row in Table IV the VOXL 2 was flown in *position* mode at a different height and heading, then held steady while the cone was placed on the floor two–to–three metres away. The pipeline still solved the planar (x, y) position with a mean error of **1.20 m**, close to the value obtained in simulation. The depth column, however, shows the small Depth-Anything model under-reads distance (e.g. 0.80 m in Fig. 6 although the tape measure read 1.5–2.0 m). We trace this bias to (i) heavier RTSP compression artefacts and (ii) the smaller network having less range accuracy. A larger checkpoint or a higher-bit-rate stream is planned for future work.

Video demonstrations. A short screen-capture of the Gazebo trial and a hand-held recording of the VOXL 2 lab test are available online:

- Simulation – autonomous fire localisation and navigation.
- Real-world – depth-assisted localisation on VOXL 2.

V. Author Contributions

Sai Jagadeesh Muralikrishnan

- *Perception* – trained the custom YOLOv8 model (VisDrone, COCO, cone data), exported a TFLite version, and built both real-world and simulation 2-D detection nodes.
- *Depth integration* – swapped the monocular-projection code for Depth-Anything V2 and benchmarked model sizes on an RTX 4070.
- *Autonomy* – refactored the off-board navigation to use 3-D centroids from depth.
- *GUI/tools* – designed and coded the PySide 6 GUI (YOLO feed, depth overlay, 3-D fire markers, person-fire alerts).

Varun Lakshmanan

- *Simulation* – created the Gazebo lab world, surrogate-fire assets, and the Phase-1 monocular pipeline.
- *Depth R&D* – compared MiDaS with Depth-Anything and tuned PyTorch on the i7-13700HX/RTX 4070.
- *Mission logic* – extended the `SEARCH→HOLD→APPROACH` state machine and trajectory generator.
- *Hardware* – set up the VOXL 2 platform, RTSP streaming, and led field tests.

Joint

- Shared system design decisions, code reviews, experiment plans, data logging, and writing of the paper.

VI. Challenges Faced

Despite the overall success of FireDroneX, several hardware–software integration hurdles required iterative troubleshooting:

- **Camera-pitch mismatch.** Our first plan—mounting the VOXL 2 front camera at a $-45°$ tilt to mimic the Gazebo setup—produced severe vibration and image blur. We reverted to the native forward-facing orientation (pitch $0°$) and compensated in software.
- **PX4 local-pose instability after SDK 1.14 reflash.** Loading the recommended parameter set (`m500_FCV2.params`, `indoor_vio.params`, `voxl2_io_enable_pwm.params`) still yielded drifting odometry. Stable VIO was achieved only after adding a missing "indoor VIO (no GPS)" helper file sourced from the ModalAI forum, occasionally followed by `systemctl restart voxl-px4` or a full reflash.
- **Intermittent ADB/USB link.** Certain USB-C cables appeared healthy but failed to enumerate the device after a few minutes. Swapping to higher-quality cables restored ADB; the fault recurred with the original leads.
- **Unreliable RTSP video feed.** The default `voxl-streamer` service provides only one low-bandwidth H.264 stream; opening it simultaneously in QGroundControl and the GUI caused frame drops. Editing `/etc/modalai/voxl-streamer.conf` to enable `hires_small_color_encoded` and a second stream improved resilience, but the bit-rate remains a bottleneck for high-resolution inference.
- **Depth-estimation model sizing.** Depth-Anything "large" exceeded VRAM on the drone and stuttered on the ground station at >30Hz. We down-selected to the "base" checkpoint to maintain real-time throughput on the RTX 4070.
- **ROS 2 QoS mismatch.** Some VOXL 2 topics were published with `RELIABLE` QoS, while our ground-station nodes used `BEST_EFFORT`. The mismatch blocked depth and detection feeds at random. For every topic we forced both sides to `BEST_EFFORT`; after that the links came up quickly and stayed stable.

Most issues were mitigated by tighter parameter management, better cabling, and selective model scaling; however, stream bandwidth remains a topic for future work (e.g., hardware H.265 encoding or Wi-Fi 6E).

VII. Conclusion

FireDroneX shows that one normal RGB camera, helped by new deep-learning models, can give useful 3-D fire and person data without the weight or cost of LiDAR or stereo sensors. A custom YOLOv8n detector (78 precision, 64 % recall) finds fires and people; Depth-Anything V2 turns these 2-D boxes into world points, driving a simple `SEARCH{HOLD{APPROACH` loop that guides the drone to within **1.2 m** of the true fire spot in simulation. Tests on a VOXL 2 air-frame confirm clear depth read-outs and live GUI views of targets and flight path.

Main lessons learned:

- Large depth networks can match short-range active sensors if the camera is well calibrated.
- Holding for just 2–3 frames lets depth values settle before the drone moves in.
- A light GUI that mixes video, depth, and state info helps operators act quickly.

References

[1] J. Gaigalas, L. Perkauskas, H. Gricius, T. Kanapickas, and A. Krisciunas, "A Framework for Autonomous UAV Navigation Based on Monocular Depth Estimation," *Drones*, vol. 9, no. 4, Art. no. 236, 2025, doi: 10.3390/drones9040236. [Online]. Available: https://www.mdpi.com/2504-446X/9/4/236

[2] M. Stephan, B. Battseren, and W. Hardt, "Object Localization in 3D Space for UAV Flight Using a Monocular Camera," M.S. thesis, Chemnitz Univ. of Technology, Chemnitz, Germany, 2021. [Online]. Available: https://www.tu-chemnitz.de/informatik/service/ib/pdf/CSR-21-01.pdf

[3] Y. Hu, S. Fang, W. Xie, and S. Chen, "Aerial Monocular 3D Object Detection," *arXiv preprint* arXiv:2208.03974, 2022. [Online]. Available: https://arxiv.org/abs/2208.03974

[4] L. Yang *et al.*, "Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2024. [Online]. Available: https://github.com/LiheYoung/Depth-Anything

[5] P. Zhu *et al.*, "Vision Meets Drones: A Challenge," in *Proc. Eur. Conf. Computer Vision (ECCV) Workshops*, Munich, Germany, 2018. [Online]. Available: http://www.aiskyeye.com/

[6] Y. Xiao and N. Di, "SOD-YOLO: A Lightweight Small Object Detection Framework," *Scientific Reports*, vol. 14, Art. no. 25624, 2024, doi: 10.1038/s41598-024-77513-4.

[7] PX4 Autopilot Dev Team, "PX4 Developer Guide," Online manual, 2023. [Online]. Available: https://docs.px4.io/

[8] ModalAI Community, "VOXL 2 indoor VIO parameter set (forum thread)," ModalAI Forum, Apr 2024. [Online]. Available: https://forum.modalai.com/

[9] OpenAI, "ChatGPT large language model," version o3, May 2025. [Online]. Available: https://chat.openai.com/