PX4 AI Tracker: Autonomous Drone System with Computer Vision and Thermal Tracking

Ayachi Dhia Eddine[a,1], **Zemmouli Ahmed**[b,1] **and Touhari Mounir**[c,1]

[a]*Faculty of Electrical Engineering, USTHB*

Master SIPNOM – Intelligent Systems

Abstract—This report presents the design, implementation and testing of **PX4 AI Tracker**, an autonomous drone system built on PX4 Autopilot, ROS 2 Humble and Gazebo Classic. The project integrates real-time computer vision with flight control to achieve two main capabilities: (1) autonomous road following over a realistic city environment using OpenCV-based lane detection and PID lateral control, and (2) nighttime thermal human tracking with PID visual servoing. A graphical control interface provides live camera feed, telemetry display and mission selection. The full software stack — from Micro XRCE-DDS bridging to offboard trajectory control — is described, together with the custom Gazebo worlds, drone models and launch infrastructure. Results from simulation runs demonstrate successful autonomous navigation, target acquisition and multi-mission operation including circle orbits, figure-8 patterns, grid scans and return-to-home.

Keywords—*PX4, ROS 2, Gazebo, autonomous drone, computer vision, OpenCV, thermal tracking, PID control, offboard mode, SITL simulation*

1. Introduction

Unmanned aerial vehicles (UAVs) are increasingly used in surveillance, search and rescue, infrastructure inspection and precision agriculture. These applications require the drone to operate *autonomously*: perceiving its environment through on-board sensors, making decisions in real time and controlling its own flight path without continuous human input.

The **PX4 AI Tracker** project demonstrates a complete autonomous drone pipeline built entirely in simulation. The system combines:

- the **PX4 Autopilot** flight controller running in Software-In-The-Loop (SITL) mode [9], [10],
- the **Gazebo Classic** physics simulator with custom 3-D worlds [6],
- **ROS 2 Humble** as the middleware for inter-process communication [4],
- **Micro XRCE-DDS** to bridge PX4's internal uORB bus to the ROS 2 network [5],
- **OpenCV** for real-time image processing and computer vision [7].

Two main autonomous missions are implemented:

1. **City Road Tracker** — the drone flies over a photo-realistic city (CitySim [2]) and follows road lines detected by a downward-facing camera, using HSV colour segmentation and contour analysis.
2. **Nighttime Thermal Tracker** — the drone searches for and tracks human heat signatures in a dark environment using a simulated thermal camera and PID visual servoing.

In addition, a **Drone Control GUI** (Tkinter) provides live video, telemetry and one-click access to nine different flight missions.

This report covers the full development process: environment setup, architecture design, implementation details of every node, testing methodology and results.

2. System Architecture

The overall system consists of five major layers, as shown in Figure 1.

2.1. PX4 Autopilot (SITL)

PX4 is an open-source flight controller used on real hardware and in simulation [9]. In SITL mode, PX4 runs as a native process and communicates with the simulator through shared memory. It provides *offboard mode*, which allows an external computer (or ROS node) to send position and velocity setpoints at 2 Hz or faster [8]. All our controllers use this mode.

2.2. Gazebo Classic

Gazebo provides the 3-D physics engine, sensor simulation and rendering. We created or adapted three custom worlds:

- `road_cars.world` — a city environment with roads, buildings and moving cars, used for the road-following mission.
- `city_drone.world` — a variant with the CitySim city model for visual-only perception.
- `nighttime_thermal.world` — a dark scene with emissive human models for the thermal tracking mission.

2.3. Micro XRCE-DDS Agent

Micro XRCE-DDS bridges PX4's internal uORB messaging to the DDS domain used by ROS 2 [5]. The agent runs on UDP port 8888 and translates topics such as `/fmu/out/vehicle_status`, `/fmu/in/trajectory_setpoint` and `/fmu/in/vehicle_command` between the two systems.

2.4. ROS 2 Humble

All perception and control nodes are standard ROS 2 Python nodes communicating through topics. The QoS profiles are configured for best-effort reliability and transient-local durability to match PX4's defaults [4].

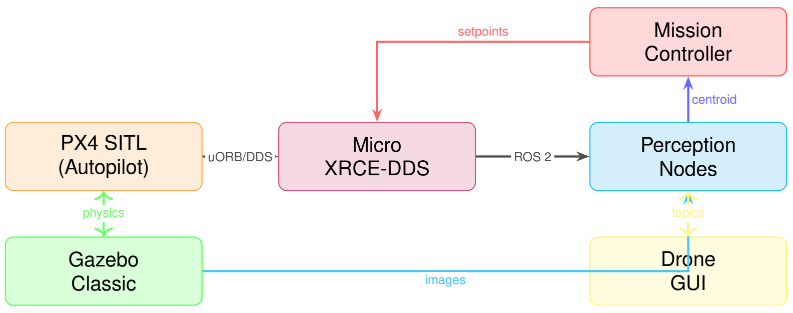

Figure 1. High-level system architecture of PX4 AI Tracker.

3. Environment Setup and Installation

The full installation was carried out on **Ubuntu 22.04 LTS** [3]. The following subsections document every step.

3.1. Prerequisites

1. **Ubuntu 22.04** — installed as the base operating system (either bare-metal or dual-boot).
2. **ROS 2 Humble** — installed from the official APT repository following the Debian package method [4]:

```
sudo apt install ros-humble-desktop
source /opt/ros/humble/setup.bash
```

3. **Python 3 dependencies** installed via `pip`:

```
pip3 install opencv-python numpy \
     Pillow mavsdk
```

4. **Gazebo Classic 11** — installed alongside ROS:

```
sudo apt install gazebo ros-humble-gazebo-*
```

3.2. PX4 Autopilot

The PX4 firmware was cloned and built for SITL:

```
git clone https://github.com/PX4/PX4-Autopilot.git \
    --recursive
cd PX4-Autopilot
bash Tools/setup/ubuntu.sh
make px4_sitl gazebo-classic
```

This compiles the autopilot firmware and the Gazebo Classic plugin, and verifies that a default empty world starts correctly.

3.3. Micro XRCE-DDS Agent

The DDS bridge is required for ROS 2 to communicate with PX4:

```
git clone https://github.com/eProsima/\
    Micro-XRCE-DDS-Agent.git
cd Micro-XRCE-DDS-Agent && mkdir build
cd build && cmake .. && make -j$(nproc)
sudo make install
sudo ldconfig
```

3.4. ROS 2 workspace and `px4_msgs`

A colcon workspace was created with the PX4 message definitions:

```
mkdir -p ~/ros2_ws/src && cd ~/ros2_ws/src
git clone https://github.com/PX4/px4_msgs.git
cd ~/ros2_ws && colcon build
source install/setup.bash
```

3.5. CitySim (city environment)

The CitySim package provides a realistic city with roads, buildings, sidewalks and street furniture [2]:

```
cd ~/px4_ai_tracker/citysim
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=\
    ../citysim_install
make -j$(nproc) && make install
```

The Gazebo environment variables (`GAZEBO_MODEL_PATH`, `GAZEBO_RESOURCE_PATH`, `GAZEBO_PLUGIN_PATH`) are set in the launch scripts to point to the installed CitySim assets.

3.6. Custom Gazebo Models

A custom drone model `iris_cam` was created by adding a downward-facing camera sensor to the standard Iris quadrotor. The camera plugin uses `gazebo_ros_camera` to publish images on the ROS 2 topic `/camera/image_raw`. Similarly, a `iris_thermal_cam` model was created with a thermal-style camera for the nighttime mission.

3.7. FastDDS configuration

A custom FastDDS profile (`fastdds_udp_only.xml`) was created to force UDP-only transport and avoid shared-memory issues between PX4 and the ROS 2 nodes. This file is loaded via the `FASTRTPS_DEFAULT_PROFILES_FILE` environment variable in every launch script.

3.8. Python virtual environment

A virtual environment (`venv/`) was created to isolate Python dependencies:

```
python3 -m venv venv
source venv/bin/activate
pip install opencv-python numpy Pillow
```

4. Road Detection Node

The `road_detection_node.py` is a ROS 2 node that processes the downward-facing camera feed to detect road surfaces. The processing pipeline is illustrated in Figure 2.

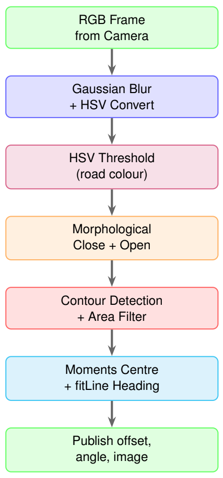
RGB Frame from Camera
Gaussian Blur + HSV Convert
HSV Threshold (road colour)
Morphological Close + Open
Contour Detection + Area Filter
Moments Centre + fitLine Heading
Publish offset, angle, image

Figure 2. Road detection processing pipeline.

4.1. HSV colour thresholding

Road surfaces (asphalt) are characterised by low saturation and medium value in the HSV colour space. The default thresholds are H in [0, 180], S in [0, 80], V in [20, 160]. These are configurable via ROS 2 parameters.

4.2. Contour analysis

After morphological cleaning, the node finds external contours and filters them by minimum area (default 500 pixels). The centre of mass is computed using OpenCV moments, and a line is fitted to the largest contour using `cv2.fitLine` to estimate the road heading angle.

4.3. Published topics

- `/road_tracker/road_offset` (`Point`):
  x = normalised lateral offset (from –1 to +1), y = road heading angle in radians, z = normalised forward offset.
- `/road_tracker/road_detected` (`Bool`).
- `/road_tracker/annotated_image` (`Image`) — debug visualisation with green road overlay, fitted line, centre marker and info text.

5. Road Follower Node

The `road_follower_node.py` is the autonomous flight controller for the city road-following mission. It implements a state machine with 14 states:

Table 1. Road Follower state machine states.

State	Description
WAITING	Idle, waiting for GUI command
PREFLIGHT	Sending offboard heartbeats
TAKEOFF	Ascending to cruise altitude
FOLLOW_ROAD	Autonomous road following
CIRCLE_AREA	Circular orbit pattern
SQUARE_PATROL	Square waypoint patrol
FIGURE8	Figure-8 (lemniscate) pattern
GRID_SCAN	Systematic grid coverage
RTH	Return to home position
FOLLOW_CAR	Follow a moving vehicle
PATROL	Road patrol with waypoints
HOVER	Static hover at position
LAND	PX4 auto-land command
DONE	Mission complete

5.1. Road following control law

The road-following mode uses the lateral offset published by the road detection node to compute yaw corrections. The commanded yaw is obtained by adding the current heading to a weighted sum of the lateral offset and the fitted road angle. Two tunable gains control how aggressively the drone corrects for lateral deviation versus aligning with the road direction. The drone moves forward at a constant cruise speed while continuously adjusting its heading to stay above the road centre line.

5.2. Mission commands

The node subscribes to `/mission/command` (`String`) and accepts commands such as `START`, `CIRCLE`, `SQUARE`, `FIGURE8`, `GRID`, `RTH`, `FOLLOWCAR`, `PATROL`, `HOVER` and `END`. These are sent by the GUI or can be published manually.

6. Thermal Perception Node

The `thermal_perception_node.py` processes a simulated thermal camera feed to detect human heat signatures in the nighttime world.

6.1. Processing pipeline

1. Convert BGR to grayscale.
2. Gaussian blur (5 x 5 kernel) for noise reduction.
3. Binary threshold at intensity 180 to isolate bright "hot" pixels.
4. Morphological open then close with an elliptical kernel.
5. Find contours, filter by area (200 to 50 000 pixels).
6. Compute bounding box and centroid of the largest contour.
7. Normalise centroid to a –1 to +1 range relative to the frame centre.

The annotated output uses the `COLORMAP_INFERNO` thermal palette with bounding boxes, centroid crosshairs and a HUD overlay showing frame count, detection status and normalised coordinates.

6.2. Published topics

- `/thermal_tracker/target_centroid` (`Point`): normalised centroid (–1 to +1 in both axes).
- `/thermal_tracker/target_detected` (`Bool`).
- `/thermal_tracker/annotated_image` (`Image`).

7. Offboard Tracker Node

The `offboard_tracker_node.py` is the mission controller for the thermal tracking scenario. It implements a state machine with seven states (PREFLIGHT, TAKEOFF, SEARCH, TRACKING, LOST_TARGET, LAND, DONE) and uses two PID controllers for visual servoing.

7.1. PID visual servoing

Two independent PID controllers adjust the drone's velocity setpoints based on the normalised centroid error [1]. The horizontal centroid error drives the lateral velocity, while the vertical centroid error drives the forward/backward velocity. The sign convention converts from image coordinates to the NED frame used by PX4. Each PID includes anti-windup clamping of the integral term and output saturation to prevent excessively aggressive corrections.

7.2. Search behaviour

When no target is detected, the drone enters SEARCH mode: it performs a slow yaw rotation to scan the area. If a target reappears within a configurable timeout, the drone switches back to TRACKING.

7.3. State transitions

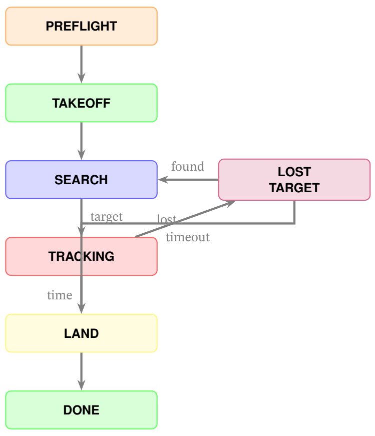

Figure 3. Thermal tracker state machine.

8. Demo Mission

The `demo_mission.py` provides a standalone demonstration of PX4 offboard control with three impressive flight patterns:

1. **Circle orbit** — 20-second circular path of radius 8 m, with tangent-aligned yaw.
2. **Figure-8** — 30-second lemniscate of Bernoulli trajectory.
3. **Spiral descent** — 20-second contracting spiral from cruise altitude down to 2 m, with decreasing radius.

The state machine handles preflight heartbeats, offboard mode activation, motor arming, autonomous flight through the three patterns, return to home position and auto-landing. The mission completes fully autonomously without any user input after launch.

9. Drone Control GUI

The `drone_gui.py` provides a full graphical control interface built with Tkinter and ROS 2. The interface features:

- **Live camera feed** — displays either the annotated road detection image or the raw camera feed, resized to fit the window.
- **Telemetry bar** — shows altitude, speed, road detection status and GPS-like position.
- **Flight controls** — ARM + TAKEOFF, LAND and emergency stop buttons.
- **Manual control** — directional pad for position, altitude (UP/-DOWN) and yaw (left/right rotation).
- **Mission selection** — nine mission buttons: Follow Roads, Follow Cars, Road Patrol, Circle Orbit, Square Patrol, Hover, Figure-8, Grid Scan and Return Home.
- **Keyboard shortcuts** — arrow keys for movement, W/X for altitude, A/D for yaw, S to start, E to land, Escape for emergency stop.

The GUI runs the ROS 2 spin loop in a separate thread while the Tkinter main loop runs in the main thread, with updates at 80 ms intervals (approximately 12 Hz).

10. Launch Infrastructure

Two automated launch scripts orchestrate the full system startup:

10.1. `launch_city_tracker.sh`

This script starts five components in sequence:

1. Set CitySim Gazebo paths (models, resources, plugins).
2. Start Micro XRCE-DDS Agent on UDP:8888.
3. Launch PX4 SITL with the `road_cars` world and `iris_cam` drone model.
4. Start the Road Detection Node.
5. Start the Road Follower Node.
6. Start the Drone Control GUI.

Each component opens in a separate terminal tab with proper ROS 2 and FastDDS environment sourcing. Waiting delays are inserted between launches to ensure clean startup ordering.

10.2. `launch_thermal_tracker.sh`

This script starts four components for the nighttime thermal mission:

1. Micro XRCE-DDS Agent.
2. PX4 SITL with the `nighttime_thermal` world and `iris_thermal_cam` model.
3. Thermal Perception Node.
4. Offboard Tracker Node (PID visual servoing controller).

10.3. `fly_now.sh`

A simpler launcher for the demo mission: starts the DDS Agent, PX4 SITL with the default empty world, and the Demo Mission node.

11. Key Design Decisions

11.1. Position-based vs. velocity-based control

The road follower uses *position-based* offboard control, sending position and yaw setpoints. This is more stable than velocity mode for waypoint following and allows PX4's internal position controller to handle the dynamics. The thermal tracker uses *velocity-based* control for the PID servoing loop, as continuous velocity corrections are more natural for target tracking.

11.2. QoS configuration

All PX4 subscribers use `BEST_EFFORT` reliability and `TRANSIENT_LOCAL` durability with history depth 1 to match PX4's internal QoS policy. Camera and inter-node topics use default QoS (reliable with depth 10).

11.3. FastDDS UDP-only transport

A custom `fastdds_udp_only.xml` profile was created to disable shared-memory transport. This avoids a known issue where PX4 and ROS 2 nodes fail to communicate when running on the same machine with shared-memory enabled.

11.4. Modular node design

Each perception and control function is a separate ROS 2 node, allowing independent testing, replacement and parallel execution. The road detection, road following, thermal perception and offboard tracking nodes can be mixed and matched for different missions.

12. Results and Discussion

The system was tested through multiple simulation runs, verifying each component and the end-to-end pipeline.

12.1. Road following performance

The road detection node successfully identifies road surfaces in the CitySim environment using HSV thresholding. The annotated camera feed shows the green road overlay, fitted centre line and offset arrows. The road follower node maintains the drone above the road centre with heading corrections at 20 Hz. The drone navigates through intersections and curved road segments autonomously.

12.2. Thermal tracking performance

In the nighttime world, the thermal perception node detects emissive human targets with the Inferno colourmap overlay. The PID visual servoing controller successfully centres the drone above the detected target, with smooth velocity corrections and low overshoot after tuning.

12.3. Demo mission

The demo mission completes all three flight patterns (circle, figure-8, spiral descent) fully autonomously. The total mission time is approximately 90 seconds, and the drone lands within 1 metre of its takeoff position.

12.4. GUI operation

The Drone Control GUI provides real-time camera feedback at approximately 10–12 frames per second. Mission switching is smooth and all nine mission modes function correctly.

13. Project File Structure

Table 2. Project files and their roles.

File	Role
`road_detection_node.py`	OpenCV road perception
`road_follower_node.py`	Road-following controller
`thermal_perception_node.py`	Thermal camera perception
`offboard_tracker_node.py`	PID thermal tracking
`demo_mission.py`	Demo flight patterns
`drone_gui.py`	Tkinter control GUI
`launch_city_tracker.sh`	City mission launcher
`launch_thermal_tracker.sh`	Thermal launcher
`fly_now.sh`	Demo launcher
`fastdds_udp_only.xml`	DDS transport config
`PX4-Autopilot/`	PX4 firmware (SITL)
`citysim/`	City simulation assets

14. Conclusion

This project demonstrates a complete autonomous drone system built from open-source components. The PX4 AI Tracker integrates:

- PX4 Autopilot for flight control with offboard mode,
- Gazebo Classic for realistic 3-D simulation,
- ROS 2 Humble for modular inter-process communication,
- OpenCV for real-time computer vision (road detection and thermal perception),
- PID control for visual servoing and path following.

The two main missions — city road following and nighttime thermal tracking — validate the end-to-end pipeline from sensor input to autonomous flight. The modular architecture allows easy extension to new sensors, control strategies or mission types. The graphical control interface provides intuitive operation for testing and demonstration.

Future work could include: integration of deep learning models (e.g. YOLO) for object detection, multi-drone coordination, real hardware deployment, and obstacle avoidance using depth sensors or LiDAR.

References

[1] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*, 1st. Princeton University Press, 2008.

[2] OSRF, *CitySim — a city simulation for gazebo*, https://github.com/osrf/citysim, 2020.

[3] Canonical, *Ubuntu 22.04 LTS (jammy jellyfish)*, https://releases.ubuntu.com/22.04/, 2022.

[4] Open Robotics, *ROS 2 Humble Hawksbill documentation*, https://docs.ros.org/en/humble/, Long-term support release, May 2022–May 2027, 2022.

[5] eProsima, *Micro XRCE-DDS agent — DDS for micro-controllers*, https://micro-xrce-dds.docs.eprosima.com/, 2024.

[6] Open Robotics, *Gazebo classic — robot simulation*, http://gazebosim.org/, 2024.

[7] OpenCV Team, *OpenCV — open source computer vision library*, https://opencv.org/, 2024.

[8] PX4 Development Team, *Offboard mode — PX4 user guide*, https://docs.px4.io/main/en/flight_modes/offboard.html, 2024.

[9] PX4 Development Team, *PX4 Autopilot — open source flight control software*, https://px4.io/, Accessed: 2025, 2024.

[10] PX4 Development Team, *PX4 SITL — software-in-the-loop simulation*, https://docs.px4.io/main/en/simulation/, 2024.