Advanced AIoT Fire Patrol Robot: Autonomous Monitoring, Interaction, and Hazard Response System

Final Project Report for YZU 114-1 EEB340A "AIoT Fundamentals"

Name: 李勇孝
Student ID Number:1144853

Abstract ***This project presents the design and implementation of an autonomous fire patrol robot system using the Robot Operating System (ROS) and the TurtleBot3 platform. Addressing the safety challenges in unmanned industrial environments, the system integrates LiDAR and RGB camera sensors to achieve autonomous navigation, real-time fire detection, and obstacle avoidance. A key innovation is the implementation of a Finite State Machine (FSM) that manages complex behaviors, including a simulated extinguishing maneuver, automated evidence logging, and a remote "escape" mechanism to prevent re-detection loops. The system was validated in a Gazebo simulation, demonstrating successful hazard identification, data recording, and human-robot interaction via remote commands.***

Keywords—ROS, TurtleBot3, Computer Vision, Sensor Fusion, Finite State Machine, AIoT.

Introduction (Heading 1)

The rapid development of Industry 4.0 has led to an increase in unmanned warehouses and automated server rooms. However, safety monitoring in these environments often relies on fixed surveillance cameras, which suffer from blind spots, or human patrols, which are costly and inefficient during night shifts. To address these issues, this project aims to develop a mobile AIoT robot capable of autonomous patrolling and active hazard response.

The objective of this project is to create a "FireBot" simulation using Gazebo. Unlike passive monitoring systems, this robot is designed to not only detect thermal anomalies (simulated by red objects) using computer vision but also to execute initial containment actions and record evidence automatically. Furthermore, the system incorporates a human-in-the-loop design, requiring operator verification before resuming patrols, thereby ensuring safety protocols are followed.

II. SYSTEM DESIGN

Hardware and Environment The system is built upon the ROS Noetic middleware running on an Ubuntu environment. The simulation platform is Gazebo, utilizing the **TurtleBot3 Waffle Pi** model. The key sensors include:

1. **LDS-01 LiDAR:** Used for 360-degree environmental scanning to detect obstacles within a 0.3-meter safety radius.
2. **RGB Camera:** Mounted on the robot to capture real-time video frames for color-based fire detection.

Software Architecture The core logic is encapsulated in a Python-based ROS node (AdvancedFireBot). The system follows a Publisher-Subscriber communication model. The robot subscribes to sensor topics (/scan, /camera/rgb/image_raw) and user commands (/control_center), processes the data, and publishes velocity commands (/cmd_vel) to the mobile base.

Finite State Machine (FSM) To manage complex behaviors, the system implements an FSM with four distinct states:

1. **PATROL:** The robot moves forward at 0.2 m/s. It utilizes sensor fusion logic where LiDAR data triggers obstacle avoidance (turning left) and visual data triggers fire detection.
2. **EXTINGUISHING:** Upon detecting a fire (red pixel count > 12,000), the robot stops and performs a "wiggling" motion for 5 seconds to simulate spraying water. Simultaneously, it saves a timestamped image and updates the log file.
3. **WAITING:** After extinguishing, the robot holds its position and waits for operator verification.
4. **ESCAPING:** Upon receiving the RESUME command, the robot executes a "blind" 180-degree turn for 3 seconds. This critical design prevents the robot from immediately re-detecting the same hazard, solving the "infinite loop" problem in simulation.

III. PROJECT OUTCOMES

A. Functional Implementation The system successfully demonstrated all proposed functions. The HSV-based vision algorithm accurately identified red objects as fire hazards. The "Sensor Fusion" logic allowed the robot to navigate around obstacles while maintaining visual scanning.

B. Data Processing and Evidence Logging A key feature of the AIoT application is data value. The system automatically generates evidence files upon hazard detection. Fig. 1 shows the system operation, where the operator view confirms the "PATROL" status, and the terminal displays the system logs.

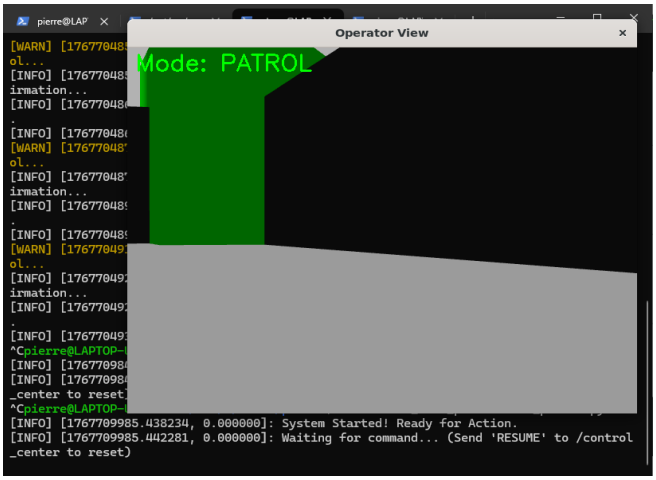

Fig. 1. System operation showing "PATROL" mode in the Operator View window and status logs in the terminal.

C. Human-Robot Interaction The integration of the /control_center topic allowed for effective remote management. The operator could successfully reset the robot's status using the RESUME command, triggering the autonomous escape maneuver.

IV. CHALLENGES AND SOLUTIONS

A. The Re-detection Loop **Challenge:** In the simulation, the "fire" object does not physically disappear after being extinguished. Consequently, when the robot resumed patrolling, it immediately re-detected the same object, causing it to get stuck in a stop-and-go loop. **Solution:** We implemented an **"ESCAPING" state**. When the resume command is received, the robot ignores visual input for 3 seconds and performs a forced rotation to face away from the hazard before reactivating its vision system.

B. Simulation Instability **Challenge:** Running Gazebo in a WSL2 (Windows Subsystem for Linux) environment caused frequent graphical crashes (Exit Code 134) due to GPU driver incompatibilities. **Solution:** We configured the environment variable export LIBGL_ALWAYS_SOFTWARE=1. This forced the system to use software rendering, which, while slightly slower, ensured a stable simulation environment for testing and recording.

C. Tuning Engagement Distance **Challenge:** Initially, the robot stopped too far from the target or mistook the fire object for a wall. **Solution:** We adjusted the visual threshold to a higher value (12,000 pixels) to ensure close-range engagement and lowered the LiDAR safety distance to 0.3 meters.

V. REFLECTIONS AND LEARNING OUTCOMES

Through this project, I gained a deep understanding of ROS communication mechanisms. One of the most valuable experiences was implementing the FSM logic. Initially, simple conditional statements led to conflicting behaviors between obstacle avoidance and fire detection. Refactoring the code into distinct states made the system robust. Additionally, implementing the automatic logging feature taught me how to bridge real-time robotics with file I/O operations, emphasizing the importance of data in AIoT applications.

VI. CONCLUSION

This project successfully developed a functional AIoT Fire Patrol Robot. By integrating ROS navigation, Computer Vision, and a remote command interface, we demonstrated a practical solution for industrial safety. The implementation of the "Escape Maneuver" and "Automated Logging" highlights the system's adaptability and practicality. Future improvements could include SLAM for map-based navigation and Deep Learning (YOLO) for more accurate object recognition.

REFERENCES

[1] R. O. S. (ROS), "Documentation - ROS Wiki," http://wiki.ros.org.
2] ROBOTIS, "TurtleBot3 E-Manual," https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.
[3] OpenCV, "OpenCV-Python Tutorials," https://docs.opencv.org/master/d6/d00/tutorial_py_root.html .

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi TIFF or EPS file, with all fonts embedded) because, in an MSW document, this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord “Format” pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.