Drone Swarm Simulation using ROS and Gazebo

Name: Naveen Kumar Reddy Basireddy
Date: 05-Nov-2025
Assignment: Round 1 – Technical Assignment (Question 3)

---

1. Introduction

In this assignment, I implemented a **drone swarm simulation** using **ROS Noetic** and **Gazebo 11** on my **Windows 11 laptop** through **WSL2 (Ubuntu 20.04)**.
The goal was to make multiple drones follow a single target based on object detection and recognition.
I designed a step-by-step workflow where drones take off to 10 meters, detect the target, approach it, and maintain a safe formation while avoiding collisions.

I divided the project into several ROS packages to handle different tasks such as model spawning, target detection, swarm coordination, and control. Each component communicates through ROS topics and services, making the whole system modular and easier to debug.

---

2. Tools and Setup

- **Operating System:** Windows 11 using WSL2 (Ubuntu 20.04)
- **Framework:** ROS Noetic with Gazebo 11
- **Programming:** Python (rospy)
- **Libraries:** OpenCV, Torch (for optional YOLO detection), and Gazebo ROS packages

To set up the environment, I installed ROS and Gazebo inside WSL, configured the DISPLAY variable to connect Gazebo GUI to Windows using **VcXsrv**, and created a **catkin workspace** with the following packages:

drone_spawn, detection_node, swarm_controller, and controller_node.

Each package has separate launch files and Python scripts, allowing me to test and modify each module independently.

---

3. Design and Working Flow

Overall Architecture

- **Gazebo world:** Contains the ground, one target object (a car model), and multiple drone models.

- **Detection Node:** Reads target position (from Gazebo model states or optional YOLO) and publishes /target/pose.
- **Swarm Controller:** Computes positions for each drone around the target using a **leader-follower** approach and a **repulsive avoidance algorithm**.
- **PID Controller:** Moves each drone smoothly toward its desired position using Gazebo's /set_model_state service.

Workflow Steps

1. All drones **take off to 10 meters**.
2. **Target detection** node publishes the position of the moving object.
3. **Swarm controller** calculates circular formation points around the target using:
4. $x = x\_target + R*\cos(\theta)$
5. $y = y\_target + R*\sin(\theta)$
6. $z = 10$
7. Drones adjust their positions continuously and maintain a **safe distance** using repulsion forces if they come too close.
8. As the target moves, the drones dynamically reposition themselves while maintaining formation.

This approach ensures that drones never collide while smoothly tracking the moving target.

---

4. Implementation Summary

Main Scripts

- **spawn_drones.py** – Spawns multiple drones and the target in Gazebo.
- **gazebo_truth_publisher.py** – Publishes the target's actual pose for simplicity.
- **swarm_controller.py** – Calculates formation points and applies collision avoidance.
- **simple_pid_controller.py** – Moves drones toward their goal using proportional control.

Launch Commands (in order)

1. roslaunch drone_spawn swarm_world.launch
2. rosrun drone_spawn spawn_drones.py _num:=4
3. roslaunch detection_node detection_multi.launch
4. roslaunch swarm_controller controller.launch num:=4

5. rosrun controller_node simple_pid_controller.py _name:=drone1
   (Repeat for other drones)

When I ran these commands, four drones took off and formed a circle around the moving target. As the target's position changed, the drones adjusted automatically. This proved that the swarm coordination logic worked correctly.

---

5. Challenges and Learnings

At first, I faced display issues in WSL because Gazebo GUI wouldn't open. After setting the DISPLAY and LIBGL_ALWAYS_INDIRECT variables, Gazebo started working properly. I also had trouble spawning multiple drones without name conflicts, which I solved by giving each one a unique namespace.

For detection, I initially planned to use **YOLOv5**, but due to GPU limitations, I switched to reading from /gazebo/model_states. This made the simulation faster and more stable.

Overall, I learned how to manage **multiple ROS nodes**, how **topics and services** connect them, and how to design **collision-free formations** in real time.

---

6. Results and Observations

- All drones successfully took off to 10 meters and followed the target in a circular formation.
- The system maintained stability even when the target changed direction.
- Distances between drones remained safe (around 2 meters).
- The behavior looked realistic and smooth in Gazebo.

---

7. Conclusion

This project helped me understand how to integrate **ROS**, **Gazebo**, and **Python control algorithms** to simulate real-world autonomous behavior.
I implemented drone spawning, detection, swarm formation, and control logic as separate ROS nodes, making the project modular and reusable.

In future versions, I plan to:

- Integrate **PX4 + MAVROS** for realistic flight control.
- Replace ground-truth detection with a **camera-based YOLO model**.
- Add obstacle avoidance and landing logic for full autonomy.

---

8. References

1. ROS Noetic Documentation – wiki.ros.org/noetic
2. Gazebo Tutorials – gazebosim.org
3. PX4 Autopilot Docs – github.com/PX4
4. YOLOv5 – ultralytics.com