 # Advanced Robotics Project: Autonomous Drone Navigation and Object Detection

Project github and installation instructions:
https://github.com/MuradMu/advanced_robotics_project/tree/main

Students:

1. Murad Murad, 324994219

2. Jwana Abuleil, 318696317

1. Introduction

The Advanced Robotics Project focuses on developing an autonomous drone capable of navigating a simulated environment, identifying a predefined object, landing on it, and returning to its initial position. The system is implemented using **ROS2**, **Gazebo**, **ArduPilot**, and **YOLOv7**, ensuring seamless simulation.

---

2. System Architecture

2.1 Software Stack

- **ROS2**: Middleware for managing robotic control and communication.
- **Gazebo**: 3D physics simulation for testing flight dynamics.
- **ArduPilot**: Open-source autopilot software for controlling UAV behavior.
- **YOLOv7**: Real-time object detection framework (for detecting the chair and other objects).
- **MAVLink & DroneKit**: Used for vehicle communication and mission execution.

2.2 Workflow Overview

1. The **simulation is launched** using iris.launch.py, which spawns the drone in Gazebo.
2. The **drone is connected** to ArduPilot for flight control using ardupilot.py.
3. **Odometry data** is published by odom_publisher.py, ensuring real-time position tracking.
4. **Transformations** between coordinate frames are handled by dynamic_tf.py.
5. The **autonomous mission** is controlled by mission.py and real_time_node.py, which guide the drone towards a detected object, align it, and execute landing.
6. The drone completes the mission and **returns to its initial location**.
7. Testing and validation of different mission parameters are performed using test.py.

---

3. Implementation Details

3.1 Launching the Simulation

Key File: **iris.launch.py**

This launch file initializes the simulation environment, ensuring the drone spawns correctly in Gazebo. The important elements include:

- Setting the **Gazebo world** file.
- Defining the **SDF model** for the drone.
- Launching **Gazebo server and client**.
- Spawning the **ArduPilot-controlled drone** in the simulation.
- Initializing **TF publishers and odometry tracking**.

Key File: **iris_publisher.launch.py**

This file ensures that the drone's **URDF model** is published to ROS2, enabling proper visualization and kinematic representation.

3.2 ArduPilot Integration

Key File: **ardupilot.py**

This script runs the ArduPilot simulation using the command:

```
sim_vehicle.py -v ArduCopter -f gazebo-iris --console
```

It starts the **Software-in-the-Loop (SITL)** simulation, allowing the drone to be controlled via MAVLink.

3.3 Odometry and Transform Handling

Key File: **odom_publisher.py**

- Subscribes to /model_states from Gazebo to get the drone's position.
- Publishes odometry data on /odom, enabling navigation control.
- Uses TF broadcaster to maintain correct coordinate transformations.

Key File: **dynamic_tf.py**

- Converts odometry data into TF transformations.
- Maintains correct relative positioning between odom, base_link, and base_scan.

3.4 Autonomous Navigation and Landing

Key File: **mission.py**

This is the **core mission script** that:

- Connects to the **drone using MAVLink**.
- Arms the drone and **takes off**.
- Navigates to a specified **GPS coordinate**.
- Detects the target **object (e.g., a chair)**.
- Executes **landing maneuvers** based on object alignment.
- Returns to the **home location** after task completion.

Key File: **real_time_node.py**

- Similar to mission.py, but allows **dynamic target updates**.
- Receives real-time coordinates via /destination_coordinates.

- Computes the **distance to the target** and adjusts navigation dynamically.
- Improves the drone's **adaptive behavior**.

3.5 Testing the System

Key File: **test.py**

- Provides a **simplified mission execution** to validate drone movement.
- Helps debug connectivity, takeoff, and GPS navigation.
- Ensures fundamental navigation works before running a full mission.

---

4. Results & Challenges

4.1 Results

- The drone successfully launches in **Gazebo** and follows a **predefined mission**.
- Real-time odometry tracking enables **smooth navigation**.
- The object detection module helps align the drone before landing.
- The drone **lands accurately** on the target object.

4.2 Challenges & Solutions

Challenge	Solution
Drone losing position reference	Improved TF handling using dynamic_tf.py
Object detection delays	Optimized YOLOv7 model for faster inference
MAVLink communication issues	Increased timeout and added reconnection logic
Inaccurate landing	Adjusted velocity control in mission.py

---

5. Conclusion & Future Work

Conclusion

This project successfully demonstrates **autonomous drone navigation** using ROS2, Gazebo, and ArduPilot. The integration of object detection enables precise landing, and real-time mission updates allow for dynamic adjustments.

Future Work

- **Enhance object detection** to work in diverse environments.
- Implement **real-world testing** using an actual drone.
- Optimize flight control using **PID tuning** for better stability.
- Develop **multi-drone coordination** for swarm-based applications.

---

6. References & Resources

1. **ROS2 Documentation** - https://docs.ros.org/en/rolling/
2. **Gazebo Simulation** - http://gazebosim.org/tutorials

3. **ArduPilot SITL Setup** - https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html
4. **YOLOv7 Object Detection** - https://github.com/WongKinYiu/yolov7
5. **MAVLink Protocol** - https://mavlink.io/en/
6. **DroneKit Python API** - https://dronekit-python.readthedocs.io/en/latest/
7. **TF Transformations in ROS2** - https://docs.ros.org/en/foxy/Tutorials/tf2.html
8. **Connicting ArduPilot With Gazebo** - https://github.com/ArduPilot/ardupilot_gazebo/blob/main/
9. **World And Models** - https://github.com/leonhartyao/gazebo_models_worlds_collection
10. **Autonomous Landing UAV: ROS-based autonomous landing system** - https://github.com/MikeS96/autonomous_landing_uav