Autonomous Multi-Drone Navigation System

An advanced, interactive, and beginner-friendly drone navigation system using Python, PyBullet, and computer vision for multi-drone simulation, dynamic obstacle avoidance, and real-time path planning.

Group Members-

Bankar Smitraj Computer Engineering UCS23M1009(Roll no:09)

Borude Yash Computer Engineering UCS23M1020(Roll no:20)

Chaudhari Abhishek Computer Engineering UCS23M1021 (Roll no:21)

Shinde Tushar Computer Engineering UCS23M1118(Roll no:118)

Features

- **Multi-Drone Support:** Simulate and control multiple drones simultaneously, each with a unique color, independent path, and controller.
- **Dynamic Obstacles & No-Fly Zones:** Add obstacles and no-fly zones that move or animate in real time. Drones automatically replan their paths to avoid collisions and restricted areas.
- **Realistic Physics:** Accurate quadrotor dynamics and propeller visuals using PyBullet.
- **Interactive 2D Map:** Drag-and-drop UI for setting start/end points, static and dynamic obstacles, and no-fly zones. Edit the path in real time.
- **Autonomous & Manual Modes:** Switch between full autonomy (A* path following) and manual PID-based control for all drones. Manual mode uses intuitive sliders for thrust, roll, pitch, and yaw.
- **Robust Pathfinding:** A* algorithm ensures drones never cut corners or pass through obstacles/walls. Real-time replanning if the environment changes.
- **Environment Validation:** Clear errors if start/end is inside an obstacle or unreachable.
- **Smooth 3D Animation:** Animated drone movement, propeller visuals, and modern 3D UI.
- **3D Camera Controls:** Rotate, pan, zoom, and follow drones in GUI mode.

- **Mission Logging:** Every move is logged with simulated latitude/longitude and action for each drone.
- **Extensible Architecture:** Modular codebase for easy addition of new features (e.g., mission scripting, swarm behaviors, advanced planners).
- **Beginner-Friendly:** In-map tutorial overlay, clear error messages, and step-by-step guide.

Recent Updates

Drone Control System Improvements

- Enhanced quadrotor physics with proper body-frame force application
- Improved control input calculation with better position error handling
- Increased control gains and thrust for more responsive movement
- Added proper force and torque application in world frame
- Implemented better stuck detection and recovery mechanisms

Key Parameters

- Control gain (k): 2.0 for responsive movement
- Thrust: 15.0 N for better lift
- Roll/Pitch limits: ±1.0 radians
- Stuck detection threshold: 2.0 seconds
- Stuck distance threshold: 1.0 meters

Table of Contents

- [FAQ]
- [License]

Quick Start Guide

1. Clone the Repository

```bash
git clone <your-repo-url>

cd <your-repo-directory>
```

2. Install Dependencies

It is recommended to use a virtual environment:

```bash
python -m venv venv

source venv/bin/activate  # On Windows: venv\Scripts\activate

pip install -r requirements.txt
```

Main dependencies:

- numpy
- opencv-python
- pybullet

3. Run the Simulation

```bash
python src/main.py
```

- For 3D camera controls (rotate, pan, zoom), run:

```bash
```

python src/main.py --gui

```

---

# 2D Map Controls

- **Left-click:** Set the start point (green)
- **Right-click:** Add dynamic obstacle (blue)
- **Middle-click:** Add static obstacle (black)
- **Drag corners:** Resize obstacles (handles appear as red dots)
- **Left-click and drag:** Move obstacles
- **Shift+Left-click:** Mark a no-fly zone (red)
- **Drag yellow path:** Edit the path in real time
- **Right-click on a cell:** Set the end point (red)
- **Press Enter:** Confirm setup and start simulation

---

# 3D Simulation Controls

- **Switch Mode:** Press `m` to toggle between Autonomous and Manual control for all drones.
- **Manual Control:** Use on-screen sliders to adjust thrust, roll, pitch, and yaw for all drones.
- **Quit:** Press `q` to exit simulation.

## 3D Camera Controls (GUI Mode)

- **Rotate:** Left mouse drag
- **Pan:** Right mouse drag
- **Zoom:** Mouse wheel
- **Follow drone:** Press `F`

---

# Manual Mode

- When you press `m`, you enter Manual mode.
- Use the sliders in the "Parameters" window to control:

\- **Thrust:** Up/down movement (0-20 N)
```

- **Roll:** Left/right tilt (-0.5 to 0.5 rad)

- **Pitch:** Forward/backward tilt (-0.5 to 0.5 rad)

- **Yaw:** Rotation (-0.5 to 0.5 rad)

- The terminal will print which mode you are in and instructions for manual control.

Project Architecture

```
autonomous-drone-navigation/

├── src/

│   ├── control/

│   │   └── controller.py      # PID control for each drone

│   ├── perception/

│   │   └── obstacle_detector.py  # (Optional) Obstacle detection

│   ├── planning/

│   │   └── path_planner.py    # A* path planning, grid/world conversion,
obstacle management

│   ├── simulation/

│   │   ├── simulator.py       # PyBullet simulation, 3D UI, status display

│   │   ├── quadrotor.py       # Drone model, propeller visuals, physics

│   │   └── drone_fleet.py     # Multi-drone management, per-drone
paths/controllers

│   ├── ui/
```

```
│   │   └── map_ui.py          # Interactive 2D map UI, obstacle editing, path visualization

│   └── main.py                # Main entry point, scenario setup, mission logic

├── models/                    # Model files (ignored by git)

├── data/                      # Data storage (ignored by git)

├── requirements.txt           # Python dependencies

├── README.md                  # This file

├── .gitignore                 # Git ignore rules
```

File & Module Descriptions

src/main.py

- **Entry point for the simulation.** Handles the overall workflow: launches the 2D map UI, collects user input, initializes the simulation, plans paths, and manages the main simulation loop.

src/ui/map_ui.py

- **Interactive 2D map interface.** Lets users set start/end points, add and edit obstacles, dynamic obstacles, and no-fly zones. Handles all mouse interactions and draws the grid, obstacles, and instructions overlay.

src/simulation/simulator.py

- **Core 3D simulation engine.** Manages the PyBullet environment, loads the ground plane, obstacles, and drones, and handles the 3D camera view. Provides methods for stepping the simulation, rendering, and updating the status display.

src/simulation/drone_fleet.py

- **Manages multiple drones.** Keeps track of all drones, their controllers, and their assigned paths. Steps each drone in the simulation and provides access to their states.

src/simulation/quadrotor.py

- **Defines the drone model.** Handles the physical properties, propeller visuals, and application of control inputs (thrust, roll, pitch, yaw) to the drone in the PyBullet world.

src/control/controller.py

- **PID controller logic.** Computes the control inputs (thrust and torques) needed for a drone to reach a target position and velocity, using PID control.

src/planning/path_planner.py

- **Path planning and grid/world conversion.** Implements the A* algorithm for pathfinding, manages the grid representation, and converts between grid and world coordinates. Handles obstacle and no-fly zone integration into planning.

src/perception/obstacle_detector.py

- **(Optional) Obstacle detection.** Provides computer vision utilities for detecting obstacles from camera images, useful for advanced perception or research extensions.

models/

- **Drone and environment model files.** Place your custom URDF/SDF models here. This folder is ignored by git.

data/

- **Data storage.** For logs, mission data, or other outputs. This folder is ignored by git.

requirements.txt

- **Python dependencies.** List of required packages for the project.

.gitignore

- **Git ignore rules.** Prevents unnecessary or large files from being tracked by git.

README.md

- **Project documentation.** This file.

Customization & Extensibility

- **Number of Drones:**

- Set `num_drones` in `main.py` or pass as an argument to `Simulator`.

- **Drone Colors:**

- Edit the `drone_colors` list in `simulation/drone_fleet.py` for custom palettes.

- **PID Controller Gains:**

- Adjust `kp`, `ki`, `kd` in `control/controller.py` or when instantiating `Controller`.

- **Map/Grid Size:**

- Change `grid_size` and `world_size` in `planning/path_planner.py` for larger or finer maps.

- **Obstacle/Zone Behavior:**

- Modify velocity vectors in the dynamic obstacles/no-fly zones dictionaries in `main.py`.

- **Simulation Mode:**

- Use `--gui` for 3D camera controls, or default for headless mode.

- **Logging:**

- Extend the logging logic in `main.py` to export to CSV/JSON or implement mission replay.

Troubleshooting

- **Drone Not Moving:**

- Check if the drone is properly initialized with correct mass and inertia

- Verify control inputs are being applied correctly

- Ensure the path planning is generating valid paths

- Check for any collision issues with obstacles

- **Black 3D Camera View:**

- Ensure the camera is positioned above and behind the drone, looking at the scene (see `_get_camera_images` in `simulator.py`).

- **2D Map Not Visible:**

- Make sure `map_ui.draw()` is called in the simulation loop and `map_ui.close()` is only called at the end.

- **Manual Controls Not Working:**

- Press `m` to switch to Manual mode. Use the sliders in the "Parameters" window.

- **Simulation Crashes or Lags:**

- Try reducing the number of drones or obstacles. Ensure your system meets the requirements.

FAQ

Q: Can I use this as a beginner?

A: Yes! The UI is designed to be intuitive, and the code is modular and well-commented.

Q: How do I add more drones?

A: Change the `num_drones` parameter in `main.py` when creating the `Simulator`.

Q: How do I add new obstacle types or behaviors?

A: Extend the `add_obstacle` method in `simulator.py` and update the 2D map UI as needed.

Q: Can I use my own drone models?

A: Yes, place your URDF/SDF models in the `models/` directory and update the loading logic in `simulator.py`.

Q: How do I contribute?

A: Fork the repo, make your changes, and submit a pull request! See the CONTRIBUTING.md for guidelines (if available).

License

This project is licensed under the MIT License. See the LICENSE file for details.