Milestone 2: Autonomous Drone Control

1st Hongyu Li
Columbia University
hl3837@columbia.edu

2nd Zach Cox
Columbia University
zsc2107@columbia.edu

3rd John Wendlandt
Columbia University
jaw2292@columbia.edu

I. Refined Problem Statement & Updated Objectives

A central challenge for UAV autonomy is achieving robust planning and control in complex, uncertain environments. Classical strategies such as proportional–integral–derivative (PID) control [4, 2] and model predictive control (MPC) [5], combined with planners like BFS and RRT, have proven effective for trajectory tracking and static obstacle avoidance in our previous work [3] and in the Milestone 1 experiments. However, these methods exhibit clear limitations when extended to dynamic or higher-dimensional settings, where rapidly changing obstacles and sensor uncertainty demand more adaptive decision-making. Reinforcement learning approaches—particularly PPO—have shown promise in addressing these challenges [6], and recent frameworks such as NavRL demonstrate that a flat PPO policy combined with a velocity-obstacle safety shield can achieve safe navigation even in dynamic environments [7]. This motivates the exploration of RL-based navigation methods as a more flexible alternative to classical planning–control pipelines, especially when targeting realistic 3D and dynamic scenarios.

In previous reports, we explored using PID and MPC as classical baselines. However, Milestone 1 revealed several practical limitations. The PID–RRT pipeline exposed a mismatch between LiDAR-based perception and the planner's 2D geometric obstacle representation, leading to discretization errors and unreliable collision handling. These perception–control mismatches indicated that integrating MPC would require substantial additional engineering effort to stabilize, validate, and debug, despite offering limited benefit in this lightweight 2D setting. As a result, we decided to remove MPC from the comparison. Initially, we planned to debug the RRT-PID agent; however, during our debugging process we realized that the conversion of the ray-casting data for this agent would be environment specific and time-consuming. In addition, the performance of the agent was not expected to exceed the PPO agent (as seen in intial 2D results) and we wanted to focus on training our own policy. Furthermore, our proposed hierarchical reinforcement learning framework—intended to separate planning and control—was found to be ill-suited for NavRL, since both layers would operate on the same LiDAR observation space and identical action space. The hierarchical structure therefore provided no meaningful decomposition and did not justify the added complexity.

To better address the limitations observed in Milestone 1, we refine the scope of the project to focus on comparing a set of modern reinforcement learning algorithms in UAV navigation. Specifically, in addition to the baseline PPO policy used by NavRL, we introduce the Soft Actor–Critic (SAC) [1] algorithm to evaluate whether improved exploration, entropy regularization, and continuous control characteristics yield more reliable navigation behavior under identical sensing and environmental conditions. This reframing places the emphasis on algorithmic comparison within a unified environment, rather than building a new hierarchical architecture that offers no clear benefit.

Our broader goal—evaluating navigation performance in 3D dynamic environments—remains unchanged. To this end, we are setting up the 3D Isaac Sim training environment to train our SAC agent in a higher-fidelity simulator. Although we have successfully configured GPU resources, Linux desktop access, and components of the Omniverse toolchain, we have not yet been able to run the 3D Isaac Sim environment. Consequently, Milestone 2 results focus on the environment setup of NavRL in the 3D Gazebo deployment environment and the setup of our proposed SAC RL policy.

II. System Completion & Technical Details

A. Isaac Sim Training Set Up

There are the five main dependencies required for the simulation environment: Orbit (simulation framework built on Isaac Sim), OmniDrones (drone models and pre-built environments), TorchRL, TensorDict, and Warp. The overall training process works by first loading configs that specify the environment and policy type. Once the environment is instantiated, the chosen policy is trained and continuous progress is tracked via Weights & Biases.

NavRL's PPO policy is implemented as a TensorDictModuleBase, which consists of three separate modules: a feature extractor, an actor network (policy), and a critic network (value function). The overall policy takes in the sensory input, where static obstacles are represented as LiDAR images and dynamic obstacles as state vectors, as well as the robot's internal state information and outputs an action command. The feature extractor combines a CNN for handling LiDAR inputs and an MLP for dynamic objects. These concatenated inputs are then passed to a ProbablisticActor from TorchRL, which samples from a Beta distribution. Finally, the critic is a single layer MLP which predicts the state value. These components collectively define the PPO policy and are separate from the majority of the training setup.

We have begun to implement a Soft Actor-Critical (SAC) [1] policy by defining analogous modules. For the SAC policy, we are using the same feature extractor (CNN + MLP) as PPO for LiDAR, dynamic obstacles, and internal state inputs. In addition, our actor is the same as PPO and will use a Beta distribution for bounded actions. Finally, we implement the critic as two Q-value networks that take a state and action as input. For off-policy learning, we need to sample actions from the replay buffer and evaluate these specific actions. We are currently working on finishing the critic network and the communication between the actor and critic.

B. System Architecture Overview

The evaluation system architecture follows a modular robotics design with distributed ROS1 nodes handling specific responsibilities: perception processing (onboard detection), environment mapping (occupancy grid construction), navigation planning (RL policy inference), and safety monitoring (collision avoidance). This modular design enables individual components to be developed, tested, and upgraded independently while maintaining system integration through ROS topics and services. The deployment environment uses Gazebo physics simulation with ROS1 for communication between components.

C. Hardware/Software Configuration

The evaluation system uses ROS1 (Robot Operating System) and Gazebo physics simulator for deployment and testing. During deployment, the navigation system runs as ROS1 nodes that interface with the Gazebo-simulated quadcopter through standard ROS topics and services. For training, we use Isaac Sim (NVIDIA's GPU-accelerated robotics simulator) to train the reinforcement learning policy. The deployment environment specified is comparable to the training environment. The trained model is then exported as a checkpoint file and deployed in the ROS1/Gazebo environment for real-time navigation. Because the setup was quicker than the SAC agent implementation and we wanted concrete results for this milestone, we opted to first demonstrate evaluation on the trained NavRL agent.

D. Environment Setup

The Gazebo world simulation consists of a 100m × 100m ground plane with ODE physics engine (1000 Hz update rate, -9.8 m/s² gravity). Obstacles are defined as 3D models with collision geometry and exact poses. Static obstacles are randomly placed (typically in a 20m × 20m region) as cylinders (radius 0.2–0.5m, height 3.0–4.0m) and boxes (width 0.2–0.5m, height 2.0–4.0m). Dynamic obstacles are modeled using Gazebo motion plugins that move them along predefined waypoint paths at speeds ranging from 0.0–1.0 m/s.

E. Navigation Unit

The Gazebo simulation uses a quadcopter model equipped with a depth camera sensor (640×480 resolution, 30 Hz frame rate, 0.05–7.0m range, 60° horizontal field of view) that publishes color and depth images to ROS topics as the robot moves through the simulated environment.

The system handles static and dynamic obstacles through separate modules. For dynamic obstacles, the onboard detector module processes sensor information (depth images, color images, and YOLO object detection results) to extract the physical properties (position, velocity, size) of moving objects. This dynamic obstacle information is passed directly to the navigation node for real-time collision avoidance.

For static obstacles, a probabilistic 3D occupancy voxel grid is constructed from sensor data as the robot moves through the environment. The system maintains a 60m × 60m × 5m global map in memory with 0.1m voxel resolution, storing occupancy probabilities (log odds) for each voxel. Only a 5m × 5m × 5m local region around the robot's current position is actively updated in real time from depth camera images; previously mapped areas persist in memory unless explicitly cleared. The occupancy map converts depth images to 3D point clouds internally using camera intrinsics, then updates voxel occupancy probabilities based on raycasting from the camera pose. The map provides collision checking functions, synthetic LiDAR raycasting services, and static obstacle clustering in the local region, which extracts bounding boxes with position, size, and orientation. This internal representation enables path planning and collision avoidance algorithms to navigate based on the robot's sensor-derived understanding of the environment rather than requiring direct access to the simulator's ground truth information.

The navigation node loads a pre-trained checkpoint file containing the learned policy weights, subscribes to robot odometry and goal commands from the user interface, queries the occupancy map for synthetic LiDAR scans, and queries the dynamic obstacle detector for moving object information (position, velocity, size). The node processes these sensor inputs through the trained neural network policy to generate velocity commands. These commands are then passed through a safety module that applies additional collision avoidance constraints using ORCA (Optimal Reciprocal Collision Avoidance) before publishing control commands to the robot. The complete system integrates perception (mapping and detection), planning (RL policy inference), and control (command execution) to enable safe navigation to goal positions while avoiding both static obstacles in the occupancy grid and dynamic obstacles tracked by the detection system.

III. Experimental Results & Intermediate Evaluation

A. Experiment Overview

We set up experiments to run the drone in 2D and 3D environments. The drone spawns at a set coordinate, and target coordinates randomly spawn in the environment. The drone navigates through the environment to reach the target. We use several metrics (defined below) to evaluate the performance of the control and planned trajectory. The primary goal is to reach the target. We use a distance threshold to define success—if the drone reaches within the threshold distance, it is considered successful. Additionally, we record the time taken to reach the target. If the drone does not reach the goal within the timeout period, flies off the map boundaries, or crashes, the episode is defined as a failure.

B. Metrics Definition

1) **Goal Reaching:** Success is defined when the drone reaches within 0.3m of the target position. When the distance to goal is less than 0.3m, the episode terminates successfully.

2) **Collision Rate:** The system records the number of collisions between the drone and obstacles. A collision is detected when the robot center (radius 0.3m) intersects with an obstacle. Due to the physics simulation, the drone may recover from collisions and continue navigating, so we track the total collision count throughout the episode rather than immediately terminating.

3) **Failure Cases:**

- **Crash:** If the drone crashes into an obstacle and becomes stuck or stops flying, the episode is terminated as a failure.
- **Timeout:** If the drone does not reach the goal within 60 seconds (600 frames at 0.1s timestep), the episode is terminated as a failure.
- **Fly Off Map:** If the drone exits the map boundaries (40m × 40m for 2D experiments, 60m × 60m × 5m for ROS1 deployment), the episode is terminated as a failure.

C. 2D NavRL Results

In the 2D NavRL environment (40×40 grid, 0.5 unit resolution, disc robot of radius 0.3, 300-step horizon), the PPO-based NavRL agent provides an initial baseline estimate. Across 30 episodes with randomly sampled start/goal cells and static obstacles, the agent reached the goal in 29 trials (96.67% success), confirming reliable point-to-point navigation under local inputs and modest environmental variability. Collisions were relatively infrequent, occurring in only 7 of the 29 successful runs. The mean ± std values below summarize collisions across all episodes, those that reached the goal, and those that did not.

TABLE I
NavRL 2D Summary Metrics

Metric	**Result**
Reached goal	29/30 (96.67%)
Collisions (all)	2.20 ± 4.26 (N = 30)
Collisions (reached_goal)	2.03 ± 4.24 (N = 29 (96.67%) | had_collision = 7)
Collisions (not_reached)	7.00 ± nan (N = 1 (3.33%) | had_collision = 1)

D. 3D NavRL Results

In the 3D Gazebo environment, we successfully ran one preset example with the NavRL agent. The run was successful and the drone reached the goal within 0.3m (the stopping threshold in ROS1 deployment). However, we were unable to rerun the examples on new virtual machines and are still configuring the new environment to scale the experiment runs and compare results with 2D environment results.

Additionally, we have multiple environment configurations that can be explored further. The default world generator configuration uses 60 static obstacles (30 cylinders and 30 boxes) and 40 dynamic obstacles (20 cylinders and 20 boxes) distributed in a 20m × 20m region, with dynamic obstacle velocities ranging from 0.0–1.0 m/s. Alternative environment setups can be configured for different difficulty levels:

- **Open Environment:** Minimal static obstacles (10–15 randomly placed cylinders and boxes), no dynamic obstacles.
- **Moderate Environment:** Moderate obstacle density (20–25 static obstacles) with few moving obstacles (1–3 objects moving at 0.5–1.0 m/s).
- **Dense Environment:** High obstacle density (40+ static obstacles) and multiple dynamic obstacles (3–5 objects moving at 0.2–1.0 m/s) along randomized waypoint paths.

IV. Final Plan & Remaining Work

A. Final Report: Dynamic Environments and Reactive Baselines (Due by Dec. 15)

- **Objectives:** Train the SAC policy and evaluate in the 3D dynamic environment. Compare the performance of a SAC agent to the baseline PPO agent.
- **Tasks:**
  - Complete SAC policy implementation and test training for a single epoch using the Issac Sim setup provided by NavRL.
  - Train the full SAC policy.
  - Create 3D Gazebo environment setups (static, dynamic, mixed) to evaluate NavRL and SAC agent's on.
  - Evaluate the policy's performance in the 3D Gazebo static environment and compare to PPO (NavRL).
  - Evaluate the policy's performance in the 3D Gazebo dynamic environment and compare to PPO.
  - Conduct ablation studies on noise robustness, latency tolerance, and policy generalization.
- **Metrics:** dynamic success rate, number of collisions, time-to-goal, and generalization to unseen obstacle setups.
- **Risk Mitigation:** While we complete the implementation of the SAC policy and training, we will implement the Gazebo environments in parallel. Further, we will collect performance metrics for NavRL. When training is done, we will have ready-to-go environment setups to collect performance metrics for the SAC policy. If we encounter training instability or planner debugging delays, we may not have time to perform ablation studies. In that case, we will only compare the performance of our policy directly to NavRL in the environment setups.
- **Expected Deliverables:** Comprehensive performance analysis across static and dynamic environments, including comparative metrics, visual trajectory results, and final discussion on the advantages of PPO and SAC policies for UAV navigation.

References

[1] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

[2] H. Hans, Y. Ma, and T. Ohtsuka. Review of pid controller applications for uavs. *arXiv preprint arXiv:2311.06809*, 2023. URL https://arxiv.org/abs/2311.06809.

[3] H. Li. Evaluating drone control algorithms: Final report. Technical report, Columbia University, December 2024. Course Project Report.

[4] I. López-Sánchez and J. Moreno-Valenzuela. Pid control of quadrotor uavs: A survey. *Annual Reviews in Control*, 56:100900, 2023. doi: 10.1016/j.arcontrol.2023.100900.

[5] M. A. Sadi, A. Jamali, A. M. N. A. Kamaruddin, and S. J. Yeo. Cascade model predictive control for enhancing uav quadcopter stability and energy efficiency in wind turbulent environments. *e-Prime*, 10:100836, 2024. doi: 10.1016/j.prime.2024.100836.

[6] Z. Tan and M. Karaköse. A new approach for drone tracking using ppo-based distributed deep reinforcement learning. *SoftwareX*, 23:101497, 2023. doi: 10.1016/j.softx.2023.101497.

[7] Z. Zhou, Z. Shi, L. Zhao, C. Xiang, and Y. Xu. Navrl: Learning safe flight in dynamic environments. *arXiv preprint arXiv:2409.15634*, 2024. URL https://arxiv.org/abs/2409.15634.