UAV Obstacle Avoidance System Using Reinforcement Learning

Submitted by

Thakur Sai Madan Gopal CS22B2023
Pandiri Veeresh Kumar CS22B2026
Damptla Harsha CS22B2024

Department of Computer Science
IIITD&M Kancheepuram
May 10, 2025

Contents

1 Introduction

Unmanned Aerial Vehicles (UAVs) are becoming increasingly important in various applications including surveillance, delivery services, and search and rescue operations. One of the critical challenges in UAV deployment is autonomous navigation, particularly obstacle avoidance in complex environments. This project focuses on developing a reinforcement learning-based obstacle avoidance system for UAVs in a simulated environment.

The system enables a UAV to navigate through a 3D environment with obstacles while attempting to reach a designated target position. We explore and compare two popular reinforcement learning algorithms: Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO). The project demonstrates the application of deep reinforcement learning to solve real-world navigation problems in robotics.

2 Objective

The primary objective of this project is to develop and evaluate reinforcement learning algorithms for UAV navigation that can:

1. Successfully avoid obstacles in a 3D environment
2. Efficiently reach target destinations
3. Learn adaptive navigation policies through experience
4. Compare the effectiveness of different reinforcement learning approaches

3 Methodology

3.1 Environment Simulation

3.1.1 UAV3DEnvironment (A2C Implementation)

The A2C implementation utilizes a custom environment called UAV3DEnvironment built on PyBullet. Key features include:

1. **World Setup:**
   - 3D bounded environment with configurable dimensions
   - Obstacles represented as cuboids with random positions and sizes
   - Target represented as a green sphere
   - UAV represented as a red sphere with collision detection
2. **State Representation:**
   - 9 + N dimensional observation space (where N is the number of LIDAR rays)
   - Includes normalized position (3), velocity (3), target direction (3), and LIDAR readings (N)
   - All values normalized to [-1, 1] range for stable learning
3. **Action Space:**
   - Continuous 3D movement vector with values in [-1, 1]
   - Controls the velocity in X, Y, and Z directions
4. **Sensing:**
   - LIDAR-like ray sensing in 360° horizontal plane
   - Configurable number of rays and maximum detection distance
   - Used for detecting obstacles and environment boundaries
5. **Physics Simulation:**
   - Uses PyBullet for realistic physics
   - Collision detection between UAV, obstacles, and environment boundaries
   - Smooth velocity transitions with momentum simulation

3.1.2 ObstacleHoverAviary (PPO Implementation)

The PPO implementation builds on the gym_pybullet_drones framework, extending the HoverAviary class:

1. **World Setup:**
   - Uses the drone simulation framework with more detailed UAV physics
   - Adds obstacles and a goal position
   - Green goal sphere as target
2. **State and Action Representation:**
   - Uses kinematic observation type
   - Maps from observations to RPM-based control actions

3.2 Reward Function Design

3.2.1 A2C Reward Function

The A2C implementation uses a carefully designed reward function:

Listing 1: A2C Reward Function Components

```
# Large penalty for obstacle collisions
# Termination condition for proximity to target
# Very large reward for target collision
# Smaller approach reward for guidance
# Anti-orbiting penalty
```

Key elements:

- Strong positive reward for reaching the target
- Strong negative reward for collisions
- Small positive reward for approaching the target
- Small negative reward for proximity to obstacles
- Time penalty to encourage efficiency
- Anti-orbiting penalty to discourage circling behavior

3.2.2 PPO Reward Function

Key elements:

- Binary reward for being within goal radius
- Small progress reward based on distance
- Strong negative reward for collisions
- Altitude bounds penalty
- Proximity penalties for getting too close to obstacles

3.3 Learning Algorithms

3.3.1 Advantage Actor-Critic (A2C)

The A2C algorithm combines policy-based and value-based learning to achieve better stability and efficiency than either approach alone. This algorithm represents a synchronous variant of Asynchronous Advantage Actor-Critic (A3C).

Network Architecture:

- **Actor Network:** Maps observations to action distribution parameters (mean and standard deviation for continuous actions)
  - Input layer: State observation vector (position, velocity, target direction, LIDAR readings)
  - Hidden layers: Two fully connected layers with ReLU activations
  - Output layer: Mean action values with tanh activation and log standard deviation
  - The network outputs a multivariate normal distribution over continuous actions
- **Critic Network:** Maps observations to state value estimates
  - Similar architecture to the actor but with a single scalar output
  - Predicts the expected return (cumulative discounted reward) from each state
  - Provides a baseline for advantage estimation

Training Process:

- **Phased Curriculum Learning:**
  - Phase 1: Large target (radius 0.6) - Teaches basic navigation and collision avoidance
  - Phase 2: Medium target (radius 0.45) - Refines approach behavior
  - Phase 3: Standard target (radius 0.3) - Develops precision in target reaching
- Each phase includes up to 1000 episodes with early stopping when performance reaches threshold
- Gradual complexity increase helps the agent master fundamentals before tackling harder scenarios
- Model parameters are transferred between phases, preserving learned knowledge

Learning Updates:

- **Advantage Estimation:**
  - Computes TD-targets: $r_t + \gamma V(s_{t+1})$ for non-terminal states
  - Advantage: $A(s_t, a_t) = \text{TD-target} - V(s_t)$
  - Advantages are normalized for training stability
- **Actor Optimization:**
  - Policy gradient loss: $-\log(\pi(a_t|s_t)) * \text{Advantage}$
  - Entropy bonus: $H(\pi(\cdot|s_t))$ to encourage exploration
  - Combined actor loss: $-(\log\_probs * advantages).mean() - entropy\_coef * entropy$
- **Critic Optimization:**
  - Mean Squared Error between predicted values and TD-targets
  - $\text{MSE}(V(s_t), \text{TD-target})$
- The code uses a single backward pass with retained graph for computational efficiency

Exploration Strategy:

- **Adaptive Gaussian Noise:**
  - Noise scale: $\max(0.1, 0.5 - 0.4 * (episode/phase\_episodes))$
  - Higher noise early in training, gradually decreasing
  - Phase-dependent exploration (more in earlier phases)
- **Action Sampling and Clipping:**
  - Actions sampled from the policy distribution rather than using deterministic mean
  - Additional noise added explicitly during early training
  - Actions clipped to [-1, 1] to stay within valid action space
- This combination provides thorough exploration of the state-action space early while focusing on exploitation later

Implementation Details:

- Mini-batch Updates: Performs updates after collecting full episodes
- Parallel Sampling: The implementation could be extended to parallel environment sampling
- Momentum-Based Action Application: Actions are blended with previous velocity (0.8 * prev_vel + 0.2 * action)
- Early Stopping: Training phases end early if performance threshold is reached

3.3.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a policy gradient method that uses a clipped surrogate objective to prevent too large policy updates, maintaining stable learning. Our implementation leverages the stable-baselines3 framework.

Algorithm Fundamentals:

- **Trust Region Method:** PPO restricts policy updates to prevent performance collapse
- **Clipped Surrogate Objective:**
  - $L^{CLIP}(\theta) = \min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)$
  - Where $r_t(\theta)$ is the probability ratio $\pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$
  - This objective prevents excessively large policy updates
- **Value Function Learning:** Simultaneously learns state-value function
- **Generalized Advantage Estimation (GAE):** Uses multi-step returns with exponential weighting

Network Architecture:

- **MlpPolicy from stable-baselines3:**
  - Policy Network: Maps observations to action distribution
  - Separate networks for policy (actor) and value function (critic)
  - Default: Two shared hidden layers (64 units each) with tanh activation
  - Policy head outputs mean actions and a state-independent log std dev
- **Value Network:** Maps observations to state values
  - Same architecture as policy network but with scalar output
  - Predicts the value function $V(s)$ for advantage estimation

Hyperparameters:

- Learning Rate: 1e-4 (reduced from default 3e-4 for stability)
- N Steps: 4096 (number of steps to collect before updating)
  - Larger than default (2048) to gather more experience per update
- Batch Size: 128 (mini-batch size for updates)
- Gamma: 0.995 (discount factor, higher than usual for longer-term planning)
- GAE Lambda: 0.98 (controls the bias-variance tradeoff in advantage estimation)
- Entropy Coefficient: 0.1 (encourages exploration, higher than default)
- Clip Range: 0.2 (limits policy update size)
- N Epochs: 10 (number of passes through the data for each update)

Training Process:

- **Experience Collection:**
  - Collects 4096 steps of experience in vectorized environments
  - Computes advantages using GAE
- **Policy Update:**
  - For each epoch, randomly samples mini-batches of size 128
  - Updates policy network by maximizing clipped surrogate objective
  - Updates value network by minimizing value function loss
- **Evaluation and Checkpointing:**
  - Periodic evaluation (every 2000 steps) in separate evaluation environment
  - Saves best model based on evaluation performance
  - Uses callback for early stopping when reward threshold (200) is reached

Implementation Features:

- Vectorized Environments: Uses make_vec_env for potential parallel sampling
- Progress Tracking: Logs training metrics at regular intervals
- Model Persistence: Saves best and final models for later use
- Deterministic Evaluation: Uses deterministic action selection during evaluation

Key Differences from A2C:

- Update Mechanism: PPO uses multiple epochs over the same data with clipping
- Sample Efficiency: Generally more sample-efficient than A2C
- Stability: Typically provides more stable learning through constrained updates
- Hyperparameter Sensitivity: Less sensitive to learning rate than A2C

4 Results

4.1 A2C Performance

The A2C implementation showed successful learning through the phased training approach. The progressive reduction in target size helped the UAV learn increasingly precise navigation skills:

1. **Phase 1 (Large Target):**
    - Rapid learning to approach targets
    - Some collisions but generally successful navigation
    - Established basic navigation patterns
2. **Phase 2 (Medium Target):**
    - Refined approach behavior
    - Better obstacle avoidance
    - More consistent success rates
3. **Phase 3 (Standard Target):**
    - Precise target approach
    - Effective obstacle avoidance
    - High success rate in navigating to targets

The A2C method demonstrated good convergence properties and was able to learn effective navigation policies even with complex obstacle arrangements.

4.2 PPO Performance

The PPO implementation showed unsatisfactory performance, with the UAV frequently crashing. Several potential issues in the PPO implementation were identified:

1. **Insufficient Training Time:**
    - The code limits training to only 1e3 timesteps (local) or 1e2 timesteps (non-local)
    - This is orders of magnitude less than typically needed for PPO convergence
2. **Reward Function Issues:**
    - Binary goal reward (10.0 if within radius) creates a sparse reward problem
    - Progress reward (0.1 * (1 - distance_to_goal / 5.0)) is too small to provide meaningful guidance
    - The obstacle proximity penalty may be too weak
3. **Environment Differences:**
    - The PPO environment has more complex drone dynamics
    - Observation and action spaces differ significantly from the A2C environment
    - The control frequency and episode length may not be optimally tuned
4. **Environment Reset Issues:**
    - The obstacles list is cleared on reset but there's no verification that obstacles are properly recreated

4.3 Improving the PPO Implementation

Several improvements could be made to the PPO implementation:

1. **Extended Training Time:**
   - Increase training timesteps to at least 1e6 (one million) steps
   - Implement proper checkpointing to resume training
2. **Enhanced Reward Function:**
   - Create a continuous reward based on distance to target rather than binary
   - Increase the magnitude of the progress reward
   - Implement reward shaping to create a smoother gradient
3. **Hyperparameter Tuning:**
   - Reduce learning rate further (e.g., to 5e-5)
   - Increase batch size for more stable updates
   - Adjust entropy coefficient during training
4. **Curriculum Learning:**
   - Implement a curriculum similar to the A2C approach
   - Start with simpler scenarios and gradually increase difficulty
   - Gradually increase the number of obstacles
5. **Environment Modifications:**
   - Simplify the drone dynamics initially
   - Ensure obstacles are properly placed in consistent locations
   - Verify the observation and action space dimensions
6. **Network Architecture:**
   - Create a custom policy network rather than using the default
   - Increase network capacity with more layers/neurons
   - Consider using recurrent layers for better temporal reasoning
7. **Enhanced Debugging:**
   - Implement detailed logging of position, rewards, and actions
   - Visualize trajectory information
   - Add regression tests for environment components

5 Individual Contributions

5.1 Thakur Sai Madan Gopal (CS22B2023)

- Designed and implemented the A2C algorithm architecture including actor and critic network structures
- Developed the custom UAV3DEnvironment simulation environment with PyBullet integration
- Implemented the phased curriculum learning approach with adaptive difficulty progression
- Created the reward function engineering for the A2C implementation
- Performed hyperparameter optimization for the A2C learning process

5.2 Pandiri Veeresh Kumar (CS22B2026)

- Implemented the PPO algorithm using stable-baselines3 framework
- Developed the ObstacleHoverAviary environment based on gym_pybullet_drones
- Created the reward function for the PPO implementation
- Identified limitations in the current PPO implementation and proposed improvements
- Conducted comparative analysis between A2C and PPO performance metrics

5.3 Damptla Harsha (CS22B2024)

- Implemented the collision detection and handling system for the simulation environments
- Developed the LIDAR-based sensing capabilities for obstacle detection
- Created the momentum-based physics model for realistic UAV movement
- Optimized the environment reset mechanism and obstacle generation algorithms
- Authored significant portions of the project documentation and final report

6 Conclusion

This project demonstrates the application of reinforcement learning to UAV obstacle avoidance. The A2C implementation showed promising results with successful navigation and obstacle avoidance, while the PPO implementation requires further refinement to achieve similar performance.

Key findings include:

1. The importance of well-designed reward functions in reinforcement learning
2. The effectiveness of curriculum learning for complex tasks
3. The critical role of hyperparameter tuning in algorithm performance
4. The need for sufficient training time for policy convergence

Future work could focus on:

1. Testing the trained policies in more complex environments
2. Incorporating more advanced sensing capabilities (e.g., vision-based navigation)
3. Developing multi-agent reinforcement learning for swarm navigation

The project successfully demonstrates the potential of reinforcement learning for solving complex robotics navigation problems, particularly in the domain of UAV control and obstacle avoidance.

GitHub Repository

- Click Here