Reinforcement Learning for Air Traffic Control Approach

CS4246 Project

Felicia Scharitzer[*], A0293835R
Alejandro Trigueros Alonso[†], A0293701H
Zhang, Yang[‡], A0294105J

November 30, 2024

1 Introduction and Problem Definition

Air Traffic Control (ATC) approach management presents a complex challenge in aviation safety and efficiency. This project explores the application of Reinforcement Learning (RL) to automate and optimize aircraft approach procedures. The specific focus is on guiding aircraft through their final approach phase, where precise control and multiple constraints must be simultaneously satisfied.

1.1 Problem Statement

The fundamental challenge is to guide an aircraft from an initial approach position to its final approach fix (FAF) while satisfying multiple constraints and optimizing for efficiency. An acceptable solution must:

1. Guide the aircraft to intercept the runway centerline at an angle not exceeding 45 degrees.
2. Maintain safe altitude throughout the approach.
3. Arrive at the FAF with appropriate speed and configuration.
4. Optimize the path for fuel efficiency and time.
5. Maintain separation from terrain and obstacles.

1.2 Relevance to Reinforcement Learning

This problem is particularly well-suited to reinforcement learning for several key reasons:

1. **Sequential Decision Making:** The approach path consists of a series of interconnected decisions, where each action affects future state possibilities.
2. **Continuous State and Action Spaces:** The problem involves:
    - **State space:** Position (x, y), altitude, heading, and speed vectors.
    - **Action space:** Turn rate, horizontal acceleration, and vertical acceleration.
3. **Clear Objectives with Complex Interactions:** The problem features:
    - Well-defined success criteria (reaching FAF with correct parameters).
    - Multiple competing objectives (safety vs. efficiency).

[*]e1345606@u.nus.edu
[†]e1345472@u.nus.edu
[‡]e1345876@u.nus.edu

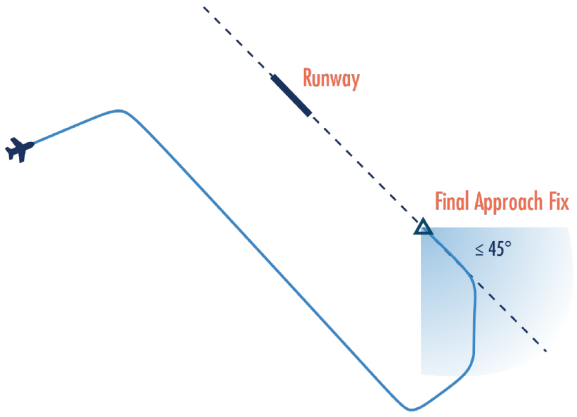


Figure 1: Illustration of the final approach challenge: guiding the aircraft to the final approach fix (FAF) with an interception angle not exceeding 45 degrees while maintaining operational constraints such as speed, altitude, and terrain clearance (1).

- Both immediate and delayed consequences of actions.

4. **Complex Constraints:** The solution must operate within strict safety and operational limits:
   - Maximum and minimum speeds.
   - Altitude restrictions.
   - Turn rate limitations.
   - Approach angle constraints.

2 Approach Environment

2.1 State and Action Spaces

The environment implements a continuous state-action space that models the essential aspects of aircraft approach control.

State Space consists of six continuous variables:

- **Position** (x, y)**:** 2D coordinates in nautical miles.
- **Heading:** Angular orientation in degrees.
- **Altitude:** Height above ground level in feet.
- **Speed:** 2D vector (horizontal and vertical components) in knots.

Action Space comprises three continuous control variables:

- **Turn rate:** Angular velocity for heading changes.
- **Horizontal acceleration:** Speed changes in the horizontal plane.
- **Vertical acceleration:** Speed changes in the vertical plane.

2.2 Environment Dynamics

The environment implements simplified aircraft dynamics where:

- Position changes are computed based on the current speed vector and heading.

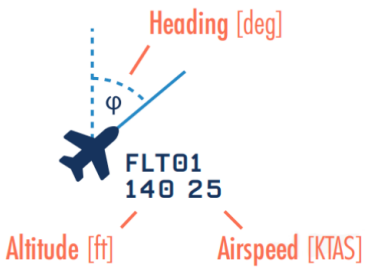


Figure 2: State space visualization showing how these variables relate to the aircraft in the environment (1).

- Speed changes are derived from applied accelerations.
- All movements respect physical constraints such as maximum turn rates and acceleration limits.

Three termination conditions are implemented:

1. Successfully reaching the target state (approach fix).
2. Crashing (violating minimum altitude or speed constraints).
3. Leaving the designated airspace.

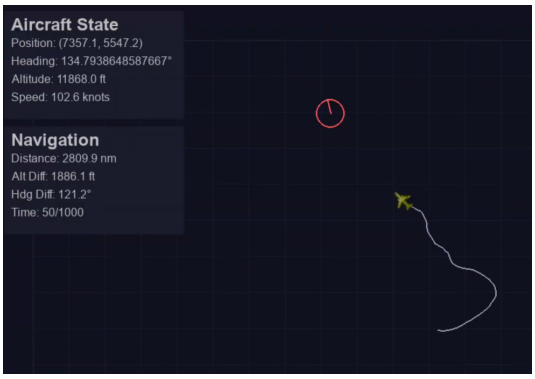


Figure 3: Screenshot of the PyGame visualization showing the environment, aircraft, and target approach fix.

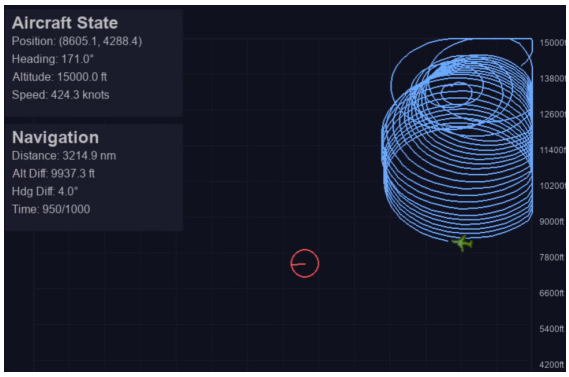


Figure 4: Visualization of the circling behavior observed with simpler reward structures.

2.3 Reward Structure

A key challenge was designing a differentiable reward function that would effectively guide learning while avoiding local optima. Early experiments revealed that simplistic reward structures led to undesirable behaviors, such as the aircraft circling indefinitely.

The reward function incorporates several components:

Positive Rewards:

- Progress toward the target position.
- Correct heading alignment.
- Appropriate altitude maintenance.
- Proper speed control.
- Proximity to the desired approach path.

Penalties:

- Time elapsed (small continuous penalty for efficiency).
- Boundary violations (smooth exponential penalty).
- Crash states.
- Airspace exits.

The implemented reward function uses normalized distances and smooth transitions to ensure differentiability. Key features include:

- Distance normalization relative to environment size.
- Progress tracking between steps.
- Smooth boundary penalties using exponential decay.
- Increasing rewards as the aircraft approaches the target.
- Balanced penalties for moving away from the target.

```
boundary_margin = self.size * 0.1  # 10% margin from edges
if dist < boundary_margin:
    penalty = np.exp((boundary_margin - dist) / boundary_margin) - 1
```

Listing 1: Example of smooth boundary penalty calculation.

This initial environment, while simplified compared to real ATC scenarios, provides a challenging continuous control problem. Despite having a known MDP (due to the absence of obstacles and other aircraft), we believed it was a useful first step in validating whether an RL agent can learn appropriate approach procedures based solely on this reward structure.

3 PPO Implementation and Mathematical Formulation

3.1 Algorithm Selection

The choice of Proximal Policy Optimization (PPO) (2) for our ATC problem was based on a careful comparison of different reinforcement learning algorithms:

Method	**Stability**	**Computational Efficiency**	**Explorability**
Deep Q-Network (DQN)	Weak in continuous scenarios	High (No 2nd-order calculation)	Weak
PPO	Strong (Clipping)	High (No 2nd-order calculation)	Strong, with entropy objective
TRPO	Strong (Trust region clipping)	Low (2nd-order calculation of trust region)	Mediocre
Soft Actor-Critic (SAC)	Strong (Entropy Regularization)	Low (Entropy calculation at each step)	Strong, with max-entropy regularization

Table 1: Comparison of reinforcement learning algorithms for continuous control tasks. (3) (4)

PPO emerged as the optimal choice for our ATC problem for several key reasons:

1. **Strong Stability in Continuous Spaces:** Unlike DQNs, which struggle with continuous action spaces (crucial for aircraft control), PPO maintains strong stability through its clipping mechanism (2).
2. **Computational Efficiency:** While TRPO and SAC offer similar stability, they require computationally expensive second-order calculations or per-step entropy computations. PPO achieves similar performance without these computational overheads (2).
3. **Strong Exploration:** PPO's entropy objective encourages robust exploration, which is essential for finding optimal approach paths in our complex state space. This matches SAC's exploration capabilities but with lower computational cost.
4. **Implementation Simplicity:** Compared to TRPO's trust region constraints or SAC's complex entropy calculations, PPO's clipping mechanism is straightforward to implement while achieving similar or better performance.

3.2 Actor-Critic Architecture

PPO uses two neural networks (2):

1. **Actor (Policy Network):**
    - Determines the agent's behavior by outputting action probabilities.
    - In our continuous action space, outputs mean and standard deviation for a Normal distribution.
2. **Critic (Value Network):**
    - Given observation as input, estimates the value function $V(s)$.

Our implementation of the PPO agent, given the extremely high complexity of the continuous ATC environment, consists of 2 around 10K parameter MLPs (Multi-Layer Perceptron). We used a PyTorch-based implementation and used ReLU as activation functions. We added a Tanh() activation function before the final output, to scale the values and prepare for action mapping.

3.3 PPO's Core Mechanism

PPO's innovation lies in its objective function (2):

$$J^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{1}$$

Where the **Probability Ratio ($r_t(\theta)$):** measures how much the current policy differs from the old one (2).

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{2}$$

However, in our case, the output action is a multi-dimensional Gaussian distribution, the similarity of which is unfit to be measured by ratios. Therefore, in our continuous implementation of PPO, we change the ratio to the KL-divergence between old and new distributions, as in

$$r_t(\theta) = D_{KL}(\pi_{old} || \pi_\theta) \tag{3}$$

Clipping (clip$(r_t(\theta), 1-\epsilon, 1+\epsilon)$): restricts the ratio to $[1-\epsilon, 1+\epsilon]$, preventing overly large policy updates (2), and **Advantage Estimation ($\hat{A}_t$):** estimates the return of an action compared to average. In our multi-dimensional action space, $\hat{A}_t$ is multi-dimensional as well.

3.4 Training Process

Being an on-policy algorithm, PPO alternates between the **Rollout** stage and the **Update** stage, where the former lets the agent play and records in the replay buffer, and the latter performs policy gradient upgrade according to memories in the replay buffer.

3.5 Implementation Optimizations

Several key features ensure stable training (2):

1. **State Normalization:**
    - Maintains running means and standard deviations of observations to ensure state dimensions are scaled comparably.
    - Crucial for scenarios where state variables like position, heading, and altitude have vastly different scales.
    - Prevents any single dimension from dominating the learning process, leading to more balanced and effective training.

2. **Parallel Environment Sampling:**
    - Multiple environments run simultaneously.
    - Improves data collection efficiency.
    - Results in better gradient estimates.
3. **Adaptive Learning Rate:**
    - Learning rate decays over time.
    - Helps fine-tune policy in later stages.
4. **Value Function Clipping:**
    - Prevents large value function updates.
    - Similar to policy clipping.

3.6 PPO's Theoretical Promise

The theoretical intuition behind choosing Proximal Policy Optimization (PPO) stemmed from several key mechanisms:

- **Controlled Policy Evolution:** The clipping mechanism aims to prevent drastic policy changes (2), which is particularly important for safety-critical domains like Air Traffic Control (ATC) where stability in continuous control is essential.
- **Architecture Benefits:** The actor-critic setup provides clearer learning signals, while the value function theoretically helps with long-term credit assignment. Parallel sampling enables better exploration of the state space (2).

However, our implementation faced significant computational challenges due to the high dimensionality of our state-action space, even for a simplified single-aircraft scenario (detailed in Section 2). Despite substantial computational resources and the 1000x speedup achieved with SB3, the agent struggled to learn efficiently.

This experience led us to reconsider our approach for a more focused application of reinforcement learning in specific scenarios, as discussed in our implementation findings (Section 4).

4 Implementation Challenges with PPO on Continuous State Space

4.1 Implementation Evolution

Our implementation progressed through two phases:

Initial Approach:
- Custom PyTorch (5) PPO + OpenAI Gymnasium (6).
- Full control but computationally intensive.
- Valuable for understanding core challenges.

Refined Approach:
- Stable-Baselines 3 (7) + OpenAI Gymnasium.
- Achieved a 1000x training speedup.
- Production-ready implementations.

4.2 Core Challenges

Two primary obstacles emerged during implementation:

1. **Broad Observation Space:**
    - **Problem:** The agent failed to learn effectively, showing high variance.
    - **Solution:** Added heuristic rewards and normalized the state space.

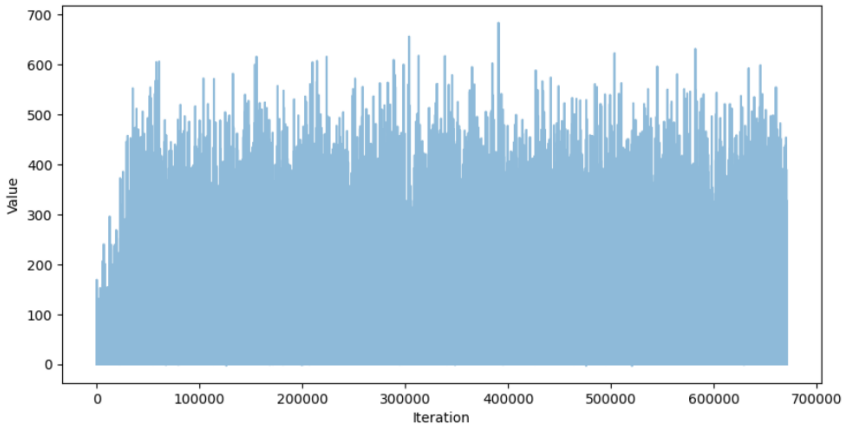


Figure 5: Initial training results showing unstable learning with a broad observation space.

2. **Non-convex Heuristics:**
    - **Problem:** Initial rewards created local optima traps.
    - **Solution:** Redesigned rewards for admissibility and smooth transitions.

4.3 Performance Analysis

Key metrics from the Stable-Baselines 3 (SB3) implementation:

- **Episode Rewards:** Steady increase from 3000 to 5000.
- **KL Divergence:** Stable at 0.005-0.01.
- **Explained Variance:** Improved from 0.65 to over 0.95.

Despite the significant training speedup with SB3, achieving convergence still required substantial computational resources for what remains a simplified scenario: a single aircraft approaching without obstacles or other traffic. This highlighted a crucial insight: when no obstacles are present, the MDP is known and can be solved deterministically!

4.4 Next Step: Hybrid System Approach

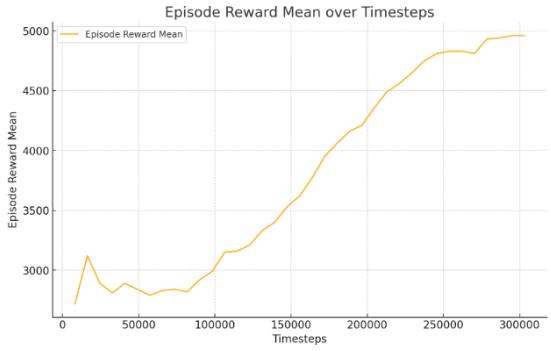
Given that the agent struggled even with the "simple" task of direct approach (where the optimal solution is known), a more efficient approach would involve a hybrid system:

- **Deterministic Solutions:** Use known MDP solutions for scenarios without obstacles or conflicts.
- **RL for Collision Avoidance:** Activate the RL agent only in scenarios where the MDP is unknown or unpredictable.

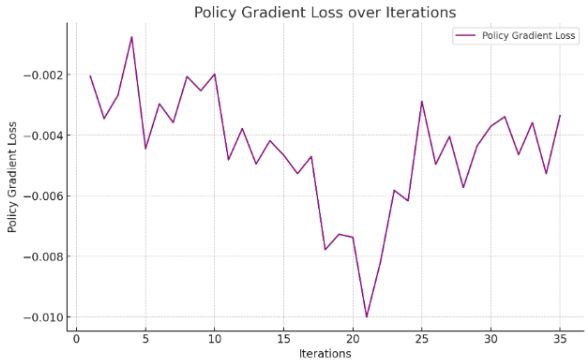
This hybrid approach leverages deterministic aspects of the system while reserving RL for situations where its flexibility and adaptability are truly needed, resulting in a more efficient and scalable solution for ATC applications.

5 Discrete Environment and Hybrid Approach

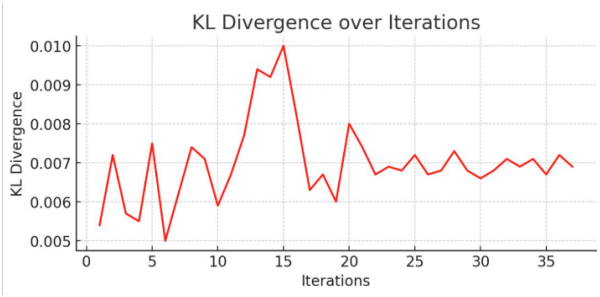
After testing our initial approach with a continuous state space, we decided on implementing the hybrid approach outlined in Section 4.4. This involved splitting the problem into deterministic path planning for simple approach scenarios and RL for unexpected obstacles and traffic. As shown in initial results in Figure 5, the size of the



(a) Episode Reward Mean over Timesteps.



(b) Policy Gradient Loss over Iterations.



(c) KL Divergence over Iterations.

(d) Explained Variance over Iterations.

Figure 6: Performance analysis metrics from the SB3 implementation: Reward Mean, Policy Gradient Loss, KL Divergence, and Explained Variance. Each metric shows the stability and progress of the training process.

continuous state space did not allow for efficient learning. This led us to redesign the environment with a simpler, discrete state space. Through discretization of the grid and actions limited to a finite set of heading changes, the state-action space was reduced to a manageable size.

5.1 SARSA in Discrete Space

As part of this shift to a discrete environment, we replaced PPO with SARSA(8) for multiple reasons:

- For the discrete action space, SARSA is sufficient in mapping states to actions, avoiding the complexity of function approximation for PPO.
- The use of an on-policy algorithm provides training stability and avoids learning from risky exploratory behavior.
- Its simple structure improves computational efficiency and debugging.

The discrete implementation simplifies the **state space** to three components:

- **Position** (x, y)**:** Grid coordinates on a 40×40 grid.
- **Heading:** Angles $\theta \in [0, 360]$ in 10 degree increments.
- **Speed:** Aircraft path progression fixed to one grid unit per time step.

The **action space** comprises five discrete actions: maintain the current heading, slight turns (±10 degrees) and hard turns (±30 degrees) in either direction.

To account for possible unexpected changes which could necessitate deviations from the deterministic path, we considered both stationary and moving obstacles. Static obstacles were placed at fixed positions on the grid, while dynamic obstacles (other planes) moved with constant speed and varying headings.

5.2 Hybrid System

The setup outlined above results in the following separation of system states:

1. Direct path planning
    - Select action that minimizes the difference in heading to the target.
    - Use geometric calculations to follow the optimal path.
2. RL with SARSA
    - Active whenever the agent enters a critical radius around an obstacle.
    - Encode distance and angle to the target and obstacles, as well as relative heading for moving obstacles.
    - Weighted rewards based on progress towards the target, heading alignment with the target and penalties in obstacle avoidance.

We automatically switch between modes based on the presence of objects in a critical distance.

5.3 Core Implementation of SARSA

SARSA (State–action–reward–state–action)(9) is typically performed by looping the following steps:

1. Choose an action based on current Q-table using ϵ-greedy policy.
2. Receive new state and reward by performing the action.
3. Update the Q value through: $Q(a_t, s_t) \leftarrow Q(a_t, s_t) + \alpha(r_t + \gamma Q(a_{t+1}, s_{t+1}) - Q(a_t, s_t)$

To improve training results, we made the following improvements:
No Q-value Updates in Deterministic Actions. As we enforce deterministic actions in obstacle-free situations, the reward information provided in these situations are often sparse or even incorrect (as slight penalties caused by obstacles from afar are ignored). Therefore, we skip the update step during the deterministic action stage and only perform the full algorithm when RL-based decisions are made.
Scheduler for Exploration Rate and Learning Rate. Although scheduling of hyper-parameters (e.g. gradually decreasing) is more common in the training of deep neural networks, we found out that incorporating such techniques does improve the result. We set the exploration rate ϵ to linearly decrease from 0.2 to 0.1, and learning rate α to linearly decrease from 0.8 to 0.2.
Intermediate Rewards. Intuitively, rewards should consist of bonuses for reaching the target, penalties for crashing into/approaching obstacles, but in practice, this set of reward functions provide too heavy a negative reward that the agent couldn't distinguish potentially beneficial actions. Therefore we added intermediate reward for approaching target and correct heading. Therefore, the reward function is:

$$R_{target} = \lambda_1(max\_dis - dis) + \lambda_2(180 - min(heading\_diff, 360 - heading\_diff)) \tag{4}$$

$$P_{obstacle} = \Sigma_i^n max(radius^2 - dis_i^2, 0) \tag{5}$$

$$R = R_{target} - P_{obstacle} + R_{terminate} \tag{6}$$

5.4 Results

A sample airport approach by our agent is shown in Figure 8. After training for 5000 episodes, the agent was able to navigate between fixed and dynamic obstacles while minimizing the deviation from the optimal path. The learning progression of the SARSA agent can be seen in Figure 7 over 5000 training episodes. In the initial phase of 0-100 episodes we can observe a spike in rewards, showing the agents quick learning of basic actions for

obstacle avoidance. The next phase shows exploration of strategies and handling of more complex situations and moving obstacles. After episode 1000, the reward stabilizes indicating that the agent converged to a reliable policy to avoid collisions and make progress towards the target.

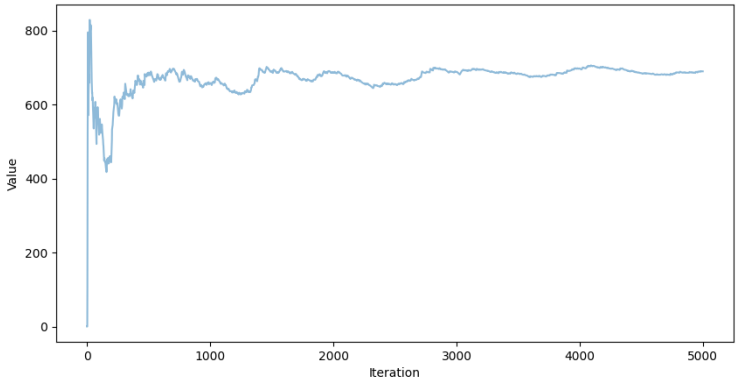
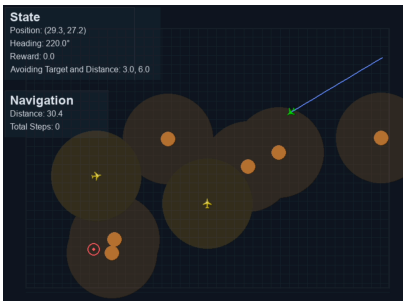


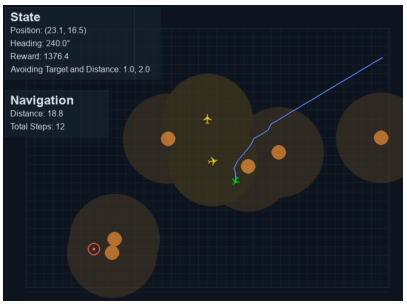
Figure 7: Mean reward across episodes.

Limitations that could be addressed in extensions of this framework include:

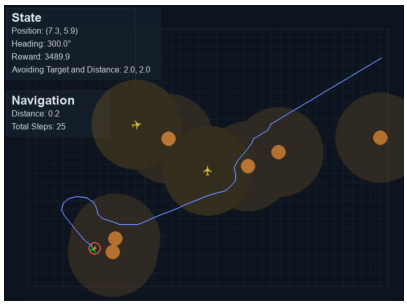
- Allowing a change in speed to optimize reaching the target.
- A more refined grid would enable more precision in navigating obstacles.
- Including a variety of obstacles and changing weather conditions would more accurately model the complexity of ATC scenarios.
- We could consider a refined reward structure including fuel efficiency, environmental impact and passenger comfort.



(a) Straight-line trajectory to airport.



(b) Within critical radius, collision avoidance with other moving planes.



(c) Rerouting to avoid stationary obstacles.

Figure 8: Frames showing an example run of navigating to an airport after SARSA training for 5000 episodes.

References

[1] Valka F. ATC Reinforcement Learning. Zenodo; 2020. Available from: https://doi.org/10.5281/zenodo.3846007.

[2] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms; 2017. Available from: https://arxiv.org/abs/1707.06347.

[3] Haarnoja T, Zhou A, Abbeel P, Levine S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In: Dy J, Krause A, editors. Proceedings of the 35th International Conference on Machine Learning. vol. 80 of Proceedings of Machine Learning Research. PMLR; 2018. p. 1861-70. Available from: https://proceedings.mlr.press/v80/haarnoja18b.html.

[4] Li Y. Deep Reinforcement Learning: An Overview; 2018. Available from: https://arxiv.org/abs/1701.07274.

[5] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al.. PyTorch: An Imperative Style, High-Performance Deep Learning Library; 2019. Available from: https://arxiv.org/abs/1912.01703.

[6] Towers M, Kwiatkowski A, Terry J, Balis JU, Cola GD, Deleu T, et al.. Gymnasium: A Standard Interface for Reinforcement Learning Environments; 2024. Available from: https://arxiv.org/abs/2407.17032.

[7] Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research. 2021;22(268):1-8. Available from: http://jmlr.org/papers/v22/20-1364.html.

[8] Rummery G, Niranjan M. On-Line Q-Learning Using Connectionist Systems. Technical Report CUED/F-INFENG/TR 166. 1994 11.

[9] Russell S, Norvig P. Artificial Intelligence: A Modern Approach. 3rd ed. USA: Prentice Hall Press; 2009.

Contributions

1. **Felicia Scharitzer**: Implementation of rendering using PyGame; Execution of experiments; Development of continuous environment, environment tailoring and debugging; Parts of presentation material; Parts of Report (Chap. 5).
2. **Alejandro Trigueros Alonso**: Project conceptualization and proposal; Environment adaptation and reward shaping for Stable-Baselines3; Integration with Stable-Baselines3 (implementing wrapper classes and vectorized environment); Development of PPO training pipeline with evaluation metrics; Parts of the Presentation; Parts of Report (Chap. 1 through 4).
3. **Zhang, Y.**: Definition of environments using OpenAI Gym; Implementation of PPO algorithm and training; Implementation of SARSA algorithm and training; Execution of experiments; Parts of presentation material; Parts of Report (3.3, 5.3).