BlueSky-Gym: Reinforcement Learning Environments for Air Traffic Applications

D.J. Groot[1] & G. Leto[1] & A. Vlaskin[1,2] & A. Moec[1], and J. Ellerbroek[1]
[1]Control and Simulation, Faculty of Aerospace Engineering, TU Delft
[2]Air Traffic Management and Airports, Royal Netherlands Aerospace Centre, NLR

***Abstract*—Reinforcement Learning (RL) is rapidly becoming a mainstay research direction within Air Traffic Management and Control (ATM/ATC). Many international consortia and individual works have explored its applicability to different ATC and U-Space / Urban Aircraft System Traffic Management (UTM) tasks, such as merging traffic flows, with varying levels of success. However, to date there is no common basis on which these RL techniques are compared, with many research parties building their own simulator and scenarios from scratch. This can diminish the value of this research, as the performance of an algorithm cannot be easily verified, or compared to that of other implementations. This hampers development in the long run. The gymnasium library shows for other research domains that this can be solved by providing a set of standardised environments, which can be used to test different algorithms, and compare them to benchmark results. This paper proposes BlueSky-Gym: a library that provides a similar set of test environments for the aviation domain, building on the existing open-source air traffic simulator BlueSky. The current BlueSky-Gym environments range from vertical descent environments, to static obstacle avoidance and traffic flow merging. Built upon the Gymnasium API and the BlueSky air traffic simulator, it delivers an open-source solution for the ATC-specific RL performance benchmark. In the initial release of BlueSky-Gym, 7 functional environments are presented. Preliminary experiments with PPO, SAC, DDPG and TD3 are presented in this paper. Results show stable training is obtained on all of the environments with the default hyperparameters. On some environments, there is a large performance gap, with the on-policy PPO often trailing, but overall no clear algorithm that outperforms others across the board in terms of total reward.**

***Keywords*—Air Traffic Management (ATM), Reinforcement Learning, Automation, Benchmarking, Artificial Intelligence**

I. Introduction

In recent years, interest for applying Deep Reinforcement Learning (DRL) methods for Air Traffic Control (ATC) tasks, such as conflict resolution and safe multi-agent navigation has grown significantly, as highlighted by the review paper on DRL applications in aviation by Razzaghi et al. [1]. However, even when just focusing on the topic of conflict resolution within ATC, the review paper by Wang et al. shows that there is a large variety in the different simulators and scenarios used for investigating the effectiveness of the different methods [2]. Even though the essence of the problems targeted in the studies are relatively similar, the parameters underlying the scenarios, such as traffic density, aircraft performance models or definition of the Markov Decision Process (MDP) often differ. This, combined with the fact that the code underlying the generated results is often not publicly available, makes it difficult to objectively compare the results of the different studies without requiring authors to reproduce all the methods.

In other domains where Machine Learning is applied, often a set of benchmark tests is used to evaluate algorithm performance, such as the MNIST dataset for image recognition [3] or the BLEU score for machine translation [4]. These standardized benchmarks improved comparability and development within their respective fields and allowed researchers to focus on methods that objectively perform better, speeding up development. For Reinforcement Learning, the solution used in several other fields is to offer simplified benchmark API's that contain a set of standardized environments, such as a the arcade learning environment [5] or the simulated road scenarios in HighwayEnv [6], which are more closely related to ATM and UTM. The most popular of these API's is The Farama Foundation's Gymnasium, which currently offers close to a hundred different environments [7]. These environments typically offer simplified scenarios, which capture the essential behaviour of the system in a more compact environment, enabling faster training and comparison of different methods than highly complex and realistic scenarios would be able to do, whilst staying true to the essence of the problem.

The Air Traffic Management research domain currently lacks such a publicly available set of environments and APIs. Fast-time open-source simulators such as BlueSky [8], AirTrafficSim [9] and Mercury [10] exist, but in order to allow for Reinforcement Learning applications, one must develop plugins from scratch every time. This slows down development and leads to the aforementioned issues of comparability of the different methods due to different implementations. Despite these challenges, BlueSky has already been used for a large number of reinforcement learning related studies ([11]–[16]).

In this paper, we present BlueSky-Gym, an open-source API based on Gymnasium and BlueSky, combined with a collection of environments. The remainder of the paper is structured as follows: the BlueSky simulator and the necessary information regarding the Gymnasium API are presented first. Then, the available environments are listed and described. An initial experiment with four RL algorithms from Stable-Baselines3 on these environments is presented in the next section. Finally, the paper ends with recommendations, further work and conclusions.

II. BLUESKY-GYM

BlueSky-Gym is built on top of the BlueSky Air Traffic Simulator [8] and inherits from the Gymnasium single agent reinforcement learning API [17], and aims to provide a standardised API, combined with benchmark enviromnents for reinforcement learning research in ATC/ATM. At the time of writing, a set of 7 baseline environments are implemented in BlueSky-Gym. It is the intention that this number will grow in the future as more environments are developed by the research community and shared through open-source initiatives.

A. BlueSky Air Traffic Simulator

BlueSky [8] is an open-source air traffic simulator developed at the TU Delft, which contains the required logic related to simulating air traffic (both conventional and UTM) while being able to do fast time simulations. Additionally, BlueSky has already been used for multiple studies investigating the efficacy of reinforcement learning for a variety of ATC/ATM tasks [11]–[16], demonstrating the applicability and versatility of using BlueSky as a platform for Reinforcement Learning research.

By default, BlueSky offers interaction with, and extension of the simulation environment through the development of plugins, which gives the user full access to all the run-time variables, allowing for tailored and detailed implementations. However, this approach of extending the functionality is not very suited in cases where there is a need for a simulation control loop external to the simulator, as is required for a large variety of open-source reinforcement learning libraries such as STABLE-BASELINES3 [18] or RLlib [19].

B. BlueSky-Gym

BlueSky-Gym aims to leverage the power of BlueSky and the ease of use of the Gymnasium API by offering a Gymnasium-derived API, which utilises BlueSky functions to simulate traffic behaviour. All of the environments developed in BlueSky-Gym inherit from Gymnasium's "Env" class, allowing direct use of standard algorithms available through reinforcement learning libraries, or custom algorithms by manually calling the "step()" function, which follows the classic Markov Decision Process (MDP) logic. To combine Gymnasium with BlueSky, different environments have been created that directly import BlueSky, and utilize its functionality for computing relevant information and handling aircraft logic and dynamics. There is therefore no need to run BlueSky on top of BlueSky-Gym.

Additionally, all environments have simplified rendering capabilities, allowing easy visualization of the learned policies. The subsequent sections will discuss the procedural generation philosophy, the open data and community-based development philosophy, potential extension to the multi-agent domain and example usage of the API.

1) Procedural Generation Philosophy

Reinforcement learning models trained on a fixed set of scenarios often exhibit overfitting, where the models memorize a sequence of actions rather than generalizing to a broader set of subtasks [20]. This is especially prevalent in environments that always have the same initial conditions such as MuJoCo and the some of the Arcade Learning environments of Gymnasium [7]. The environments in BlueSky-Gym are therefore all initialized through procedural, or random, generation, based on sets of predefined boundary conditions such as number of aircraft or airspace size. Because of this, every case encountered by the learning algorithm is different, preventing overfitting on repeated samples or memorization of a sequence of successful actions. This approach aligns with the principles of the Procgen Benchmark, a set of Gymnasium style environments that focuses on generalization in reinforcement learning [21]. Utilising procedural generation ensures that the observed rewards of the agents on the BlueSky-Gym environments can be attributed to properly trained policies.

2) Open-Data Philosophy and Community-Based Development

With the development of BlueSky-Gym we aim to foster open-data and open-source, community-based development. Open science and open-source tools are of great importance for research, and have several key advantages over closed-source alternatives. The main advantage of this philosophy is that it provides a means of benchmarking across the research community, as the need for the development of a similar platform (as well as validation against the original study) every time research is performed is eliminated. Reinforcement Learning models can then be tested by anyone using this open data and software.

Another advantage is faster prototyping, as the code is based in Python and uses open-source libraries (BlueSky Air Traffic Simulator [8] and The Farama Foundation's Gymnasium [7]), which a user can easily adjust and modify to suit their research topic. Additionally, the example environments provided for both horizontal and vertical control tasks can serve as a baseline for creating custom environments.

3) Extension to the Multi-Agent Domain

Currently the usage of BlueSky-Gym is limited to the single agent domain, having a predefined agent that is being controlled by the reinforcement learning algorithm for each episode. All other aircraft follow predefined control logic, which can be learned by the learning algorithm. However, it is acknowledged that most ATC/ATM tasks extend into the multi-agent domain. Therefore two methods for implementing multi-agent capabilities in the future are currently considered:

- **Centralized multi-agent control** - This method is similar to an air traffic controller operating on the system. Instead of modelling each agent as its own independent actor, the policy learns to control the entire joint action set of all aircraft in the system based on a global observation vector. This approach allows the usage of single agent reinforcement learning algorithms and can natively be implemented in BlueSky-Gym.
- **Distributed multi-agent control** - This turns the environments into a multi-agent reinforcement learning environments, which is currently not supported through the Gymnasium API. However, it is recognized that spe-

```python
import gymnasium as gym
import bluesky_gym
from models import your_model

bluesky_gym.register_envs()

model = your_model
env_name = "DescentEnv-v0"
training_episodes = 1e5

env = gym.make(env_name, render_mode=None)

for i in range(training_episodes):
    done = truncated = False
    obs, info = env.reset()
    while not (done or truncated):
        action = model(obs)
        obs_, reward_, done, truncated,
            info = env.step(action)
        model.train(obs, action, obs_,
            reward_)
        obs = obs_

model.save()
```

Algorithm 1. Example of manual usage of BlueSky-Gym for DescentEnv-v0.

cific APIs exist that allow for multi-agent environments such as PettingZoo [22] and MAgent2 [23]. An option is therefore to eventually maintain both a single agent branch, based on Gymnasium, and a multi-agent branch based on the aforementioned APIs.

4) API

This section demonstrates how to use BlueSky-Gym for three different use cases: training a self-defined model, training a model with the help of STABLE-BASELINES3, and testing a model trained by STABLE-BASELINES3.

Training a self-defined model requires manual interaction with the environment to obtain the relevant information. The advantage of this is more control over the learning process and used model than when relying on packages such as STABLE-BASELINES3. An example of this for one of the example environments, 'DescentEnv-v0', is given in Algorithm 1.

Algorithm 2 shows a code example for training and testing a model using the DDPG algorithm from STABLE-BASELINES3. Using libraries such as STABLE-BASELINES3 increases ease of use at the expense of control.

C. Available Environments

Seven environments are currently available within BlueSky-Gym, shown in Figure 1. Two of these are 'example' environments, created for demonstrating vertical and horizontal control logic, and functioning as a template environment for the development of new scenarios. A summary of the different environments is given in Table I. The environments are a mix of vertical and horizontal control scenarios, and are created with both a conventional ATM and a UTM perspective in mind.

```python
import gymnasium as gym
import stable_baselines3 as \textsc{\
    capsize{10}{S}table-\capsize{10}{B}
    aselines3}
import bluesky_gym


bluesky_gym.register_envs()

env_name = "DescentEnv-v0"

# Train the model
env = gym.make(env_name, render_mode=None)
model = \textsc{\capsize{10}{S}table-\
    capsize{10}{B}aselines3}.DDPG("
    MultiInputPolicy", env)
model.learn(total_timesteps=2e6)
model.save("my_model")
env.close()

# Test the model and visualize policy
env = gym.make(env_name,
    render_mode="human")
model = \textsc{\capsize{10}{S}table-\
    capsize{10}{B}aselines3}.DDPG.load("
    my_model",env=env)

done = truncated = False
obs, info = env.reset()
while not (done or truncated):
    action, _ = model.predict(obs,
        deterministic=True)
    obs, reward, done, truncated, info =
        env.step(action[()])
env.close()
```

Algorithm 2. Example of using STABLE-BASELINES3 in conjunction with BlueSky-Gym to train and evaluate a model.

1) DescentEnv-v0

DescentEnv-v0 is one of the two example environments in BlueSky-Gym. This environment is used for demonstrating vertical control logic by having the agent control the vertical speed of the aircraft, and can serve as a basis for more complex vertical control environments. The goal of the agent is to stay on a random target altitude as long as possible before descending down to the runway. The goal of the agent is therefore to learn the maximum descent rate of the aircraft model, and use that information to infer Top of Descent.

2) VerticalCREnv-v0

This vertical conflict resolution environment builds on DescentEnv-v0, and has the same structure for the target altitude and runway. However, this environment contains additional cruising aircraft with conflicting trajectories that should be avoided.

3) PlanWaypointEnv-v0

PlanWaypointEnv-v0 is the second example environment contained in BlueSky-Gym, used for demonstrating the horizontal control logic. The agent has to learn to efficiently plan a trajectory visiting randomly generated waypoints, whose

TABLE I. SUMMARY OF THE CURRENTLY AVAILABLE ENVIRONMENTS WITHIN BLUESKY-GYM.

Environment	Action Space	Observation Space	Reward	Goal
DescentEnv-v0	1, vertical speed	4	Dense	Hold target altitude, land on time
VerticalCREnv-v0	1, vertical speed	$4 + 7 \cdot n_{ac}$	Dense + Sparse	Hold target altitude, land on time, avoid conflicting aircraft
PlanWaypointEnv-v0	1, heading	$4 \cdot n_{wpt}$	Sparse	Visit all waypoints once
HorizontalCREnv-v0	1, heading	$3 + 5 \cdot n_{ac}$	Dense + Sparse	Get to waypoint, avoid conflicting aircraft
SectorCREnv-v0	2, heading & speed	$3 + 7 \cdot n_{ac}$	Dense + Sparse	Get out of sector, avoid aircraft
StaticObstacleEnv-v0	2, heading & speed	$3 + 4 \cdot n_{obs}$	Sparse	Get to waypoint, avoid static obstacles
MergeEnv-v0	2, heading & speed	$5 + 7 \cdot n_{ac}$	Dense + Sparse	Merge into traffic, avoid aircraft, reach waypoint

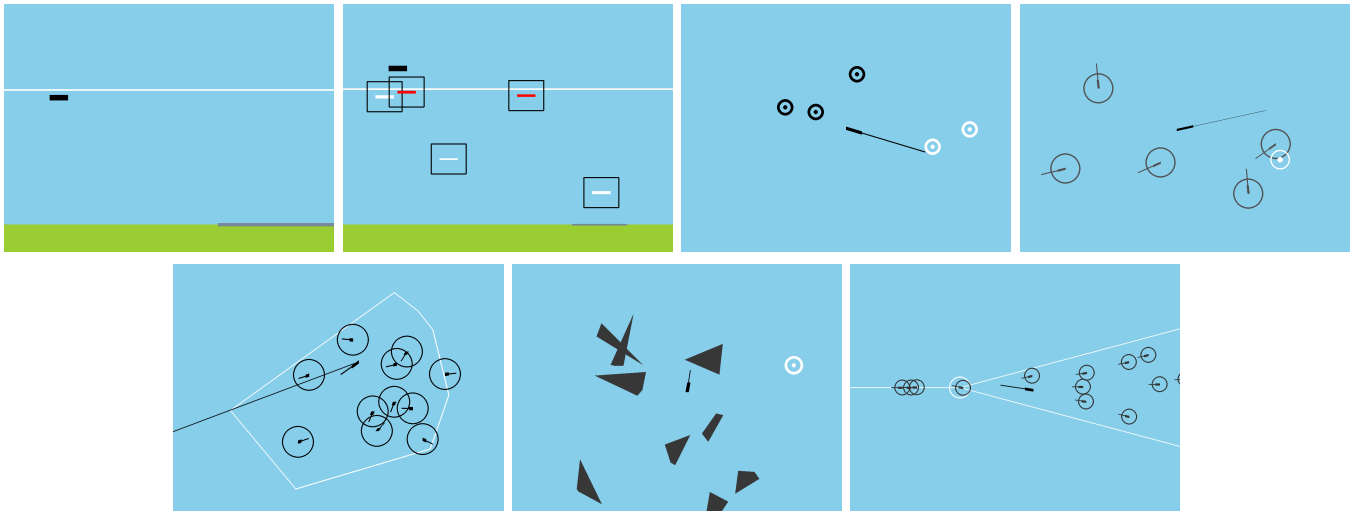

Figure 1. Visualization windows used for the different environments within BlueSky-Gym, simplified with respect to BlueSky's own GUI. From left to right, top to bottom: DescentEnv-v0; VerticalCREnv-v0; PlanWaypointEnv-v0; HorizontalCREnv-v0; SectorCREnv-v0; StaticObstacleEnv-v0; MergeEnv-v0.

location relative to the ownship is provided in the observation vector.

4) HorizontalCREnv-v0

In this environment, based on the horizontal logic of PlanWaypointEnv-v0, the agent learns to navigate to an end destination while avoiding other aircraft through heading changes. The other aircraft are initialized on a conflicting trajectory with that of the agent. Applications in ATM and UTM are conflict resolution with a destination.

5) SectorCREnv-v0

Building on HorizontalCREnv-v0, in SectorCREnv-v0 an ownship learns to exit as fast as possible the airspace sector in which it is spawned, while avoiding moving obstacles. Because of the high traffic densities and the fact that aircraft are initialized with random initial states instead of conflicting initial states like in HorizontalCREnv-v0 and VerticalCREnv-v0, this environment stimulates the training of algorithms that minimize secondary and tertiary conflicts resulting from maneuvers.

6) StaticObstacleEnv-v0

In this horizontal environment the ownship should learn to navigate to an end destination while avoiding static obstacles. This environment has concrete application both in conventional ATM and in UTM. In ATM, the static obstacles represent a simplified version of airspace restricted sectors. In UTM, they represents buildings and other physical obstacles to the navigation of drones.

7) MergeEnv-v0

The Merge environment is a horizontal scenario in which a group of aircraft are flying towards a Final Approach Fix (FAF) waypoint and continuing to a runway. This has an application in both conventional ATM and UTM, assisting landing sequencing and final approach merging. The agent controls the 'ownship', altering the speed and heading, such that the FAF is reached without conflicts.

III. INITIAL BENCHMARK EXPERIMENTS

To validate the implementations and generate initial benchmark results for the environments, a diverse set of algorithms was trained on each environment using the STABLE-BASELINES3 reinforcement learning library. These experiments are not the main subject of this paper but simply serve as a proof of concept demonstrating the functionality of BlueSky-Gym.

A. Experimental Setup

1) STABLE-BASELINES3

STABLE-BASELINES3 is an open-source library that contains stable implementations of reinforcement learning algorithms in Python using PyTorch. The library focuses on single-agent, model-free algorithms, trying to stay as true to the original implementations as possible. Because of this and the ease of implementation, STABLE-BASELINES3 was selected as the library of choice for this experiment. Nevertheless, other libraries such as RLlib or even own implementations can easily be integrated with BlueSky-Gym as well.

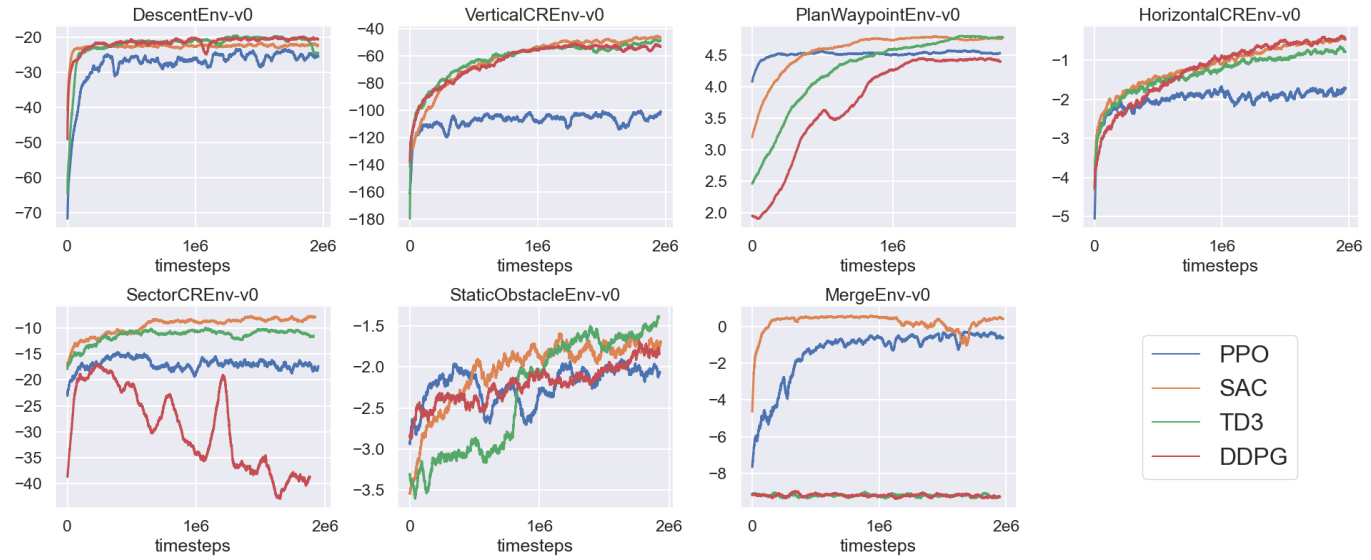

Figure 2. Evolution of the rewards for the different environments, from left to right, top to bottom: DescentEnv-v0; VerticalCREnv-v0; PlanWaypointEnv-v0; HorizontalCREnv-v0; SectorCREnv-v0; StaticObstacleEnv-v0; MergeEnv-v0.

2) Algorithms

For the experiments, a total of 4 algorithms were used to train the models on each of the environments:

- Deep Deterministic Policy Gradient (DDPG) [24]
- Twin Delayed DDPG (TD3) [25]
- Soft Actor-Critic (SAC) [26]
- Proximal Policy Optimization (PPO) [27]

Of these algorithms, three of them have been used for various RL applications in ATC/ATM [1]. However, to the best of our knowledge no study has been done using TD3 or directly comparing these different algorithms.

DDPG, TD3 and SAC are all off-policy algorithms, which implies that they utilize a memory buffer to store the samples, and are able to learn from state-transitions that have been generated by a policy that is different from the current policy.

In contrast, PPO is an on-policy method, which means that it will generate state-transitions using a fixed policy. These state-transitions are then used to update the policy directly. This strategy favours stability during learning at the cost of a lower sample efficiency.

3) Hyperparameters

For the hyperparameters, the default parameters as implemented in STABLE-BASELINES3 have been used, with an exception for the learning-rate, which was fixed at 0.0003 for all algorithms and environments. Additionally all training was done for a total of two million time-steps for all of the environments. It is acknowledged that the efficacy of reinforcement learning algorithms can be highly dependent on the selected hyperparameters. For the purpose of this study however, it was decided to use the default parameters for simplicity and ease of reproducibility of the results. Instead, the environments have been designed such that adequate results and stability can be obtained with minimal hyperparameter tuning.

B. Results & Discussion

Figure 2 shows the reward evolution for the aforementioned algorithms on the different environments.

These results indicate that all environments convey enough information to facilitate stable learning, although various differences can be observed between the different algorithms, depending on the environment. First off, there is a clear difference between PPO and the other algorithms. This can be attributed to the fact that the other algorithms are off-policy methods, relying on a memory buffer for training. Because of this, these methods have a higher sample efficiency when compared with PPO. On the other hand, PPO is more computationally efficient when performing an update step to the network, and is therefore recommended when generating samples is cheap. Because of this it is possible that two million time-steps is simply not enough for PPO to converge to a final policy, and potentially a better indicator of algorithm efficacy would be to use update steps to the networks instead of environment time-steps.

Additionally, looking at the rewards obtained by TD3 and DDPG, it is observed that for certain environments training stability seems to be an issue when compared to SAC and PPO, especially for MergeEnv-v0, where no improvements of the policies were observed at all. It is currently unknown what causes these differences, but is in line with results shown in the original SAC paper [26], where for certain environments TD3 and DDPG show no signs of improvement. One difference between TD3 and DDPG when compared to SAC and PPO is that TD3 and DDPG use deterministic policies, while SAC and PPO are stochastic. However, how these differences cause such drastic performance discrepancies is unknown.

Finally, when comparing the different reward evolutions across the environments, it is clear that a diverse set of environments have been created, as highlighted by the different reward

evolutions. One exception to this are the VerticalCREnv-v0 and HorizontalCREnv-v0 environments, which exhibit very comparable learning characteristics. This, however, is to be expected, considering that these environments are mirrors of each other in different planes of a 3D space, but with different action bounds limited by the performance models of the aircraft used.

In this experiment, we solely looked at the evolution of the rewards to verify the stability and "trainability" of the different environments. This is done, because it highlights a lot of the training characteristics of the algorithms. However, the defined reward function also directly influences learning efficacy, stability and characteristics of the final learned policy. Therefore, reward function design and tuning is often done to obtain desired behaviour. Because of this, for future experiments the reward should not be used as the key performance indicator for these different environments, as it would rule out altering the reward function as an active research topic on these environments. To accommodate this, the performance of these different algorithms on other key performance indicators, such as average deviation from the optimal track, number of intrusions, or average number of time-steps taken to reach the waypoint, amongst others, will be maintained on the GitHub page, with the intention of showcasing the best performing algorithms reported in literature.

IV. Conclusion

This paper introduced BlueSky-Gym, a platform for RL research in ATC/ATM/UTM that accommodates open-source development and benchmarking. It is based on the popular Gymnasium library and built on the open-source air traffic simulator, BlueSky. At the time of writing a set of 7 diverse environments have been developed and tested, with initial experimental results highlighting the platform's utility in benchmarking various algorithms. Nevertheless there are still some limitations to BlueSky-Gym, most notably that it currently supports exclusively single agent RL research and does not include highly complex/realistic traffic scenarios. This more simplistic approach however is also one of the main advantages of BlueSky-Gym, as it allows for easier use and comparison of different algorithms' outcomes, which would be more challenging with highly tailored complex scenarios. Although the platform is undergoing continuous improvements, it is readily available for researchers aiming to investigate RL for different challenges of Air Traffic Management.

V. Code Availability

The BlueSky-Gym environments and additional examples are available online via:
https://github.com/TUDelft-CNS-ATM/bluesky-gym

References

[1] P. Razzaghi, A. Tabrizian, W. Guo, S. Chen, A. Taye, E. Thompson, A. Bregeon, A. Baheri, and P. Wei, "A survey on reinforcement learning in aviation applications," *Engineering Applications of Artificial Intelligence*, vol. 136, p. 108911, 2024.

[2] Z. Wang, W. Pan, H. Li, X. Wang, and Q. Zuo, "Review of deep reinforcement learning approaches for conflict resolution in air traffic control," *Aerospace*, vol. 9, no. 6, p. 294, 2022.

[3] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[4] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[6] E. Leurent, "An Environment for Autonomous Driving Decision-Making," May 2018. [Online]. Available: https://github.com/eleurent/highway-env

[7] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry, "Gymnasium robotics," 2023. [Online]. Available: http://github.com/Farama-Foundation/Gymnasium-Robotics

[8] J. M. Hoekstra and J. Ellerbroek, "Bluesky ATC simulator project: an open data and open source approach," in *Proceedings of the 7th international conference on research in air transportation*, vol. 131. FAA/Eurocontrol USA/Europe, 2016, p. 132.

[9] K. Y. Hui, C. H. Nguyen, G. N. Lui, and R. P. Liem, "Airtrafficsim: An open-source web-based air traffic simulation platform." *Journal of Open Source Software*, vol. 8, no. 86, p. 4916, 2023.

[10] L. Delgado, G. Gurtner, M. Weiszer, T. Bolic, and A. Cook, "Mercury-an open-source platform for the evaluation of air transport mobility-presentation," *13th SESAR Innovation Days*, 2023.

[11] M. Brittain and P. Wei, "Autonomous air traffic controller: A deep multi-agent reinforcement learning approach," *arXiv preprint arXiv:1905.01303*, 2019.

[12] S. Deniz, Y. Wu, Y. Shi, and Z. Wang, "A reinforcement learning approach to vehicle coordination for structured advanced air mobility," *Green Energy and Intelligent Transportation*, vol. 3, no. 2, p. 100157, 2024.

[13] T. Nunes, C. Borst, E. J. van Kampen, B. Hilburn, and C. Westin, "Human-interpretable input for machine learning in tactical air traffic control," *SESAR Innovation Days 2021*, 2021.

[14] W. Guo, M. Brittain, and P. Wei, "Safety enhancement for deep reinforcement learning in autonomous separation assurance," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 348–354.

[15] D. Groot, J. Ellerbroek, and J. M. Hoekstra, "Analysis of the impact of traffic density on training of reinforcement learning based conflict resolution methods for drones," *Engineering Applications of Artificial Intelligence*, vol. 133, p. 108066, 2024.

[16] M. Ribeiro, J. Ellerbroek, and J. Hoekstra, "Distributed conflict resolution at high traffic densities with reinforcement learning," *Aerospace*, vol. 9, no. 9, p. 472, 2022.

[17] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis, "Gymnasium: A standard interface for reinforcement learning environments," 2024. [Online]. Available: https://arxiv.org/abs/2407.17032

[18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[19] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 3053–3062.

[20] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," *arXiv preprint arXiv:1804.06893*, 2018.

[21] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," in *International conference on machine learning*. PMLR, 2020, pp. 2048–2056.

[22] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 032–15 043, 2021.

[23] L. Zheng, J. Yang, H. Cai, M. Zhou, W. Zhang, J. Wang, and Y. Yu, "Magent: A many-agent reinforcement learning platform for artificial

collective intelligence," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[25] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[26] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.