A Risk-Free Approach to Validate Drone Flight Paths Using Computer Simulation and Reinforcement Learning

Dr. Md Fashiar Rahman, The University of Texas at El Paso

Dr. Md Fashiar Rahman is an Assistant Professor of the Industrial, Manufacturing and Systems Engineering (IMSE) Department at The University of Texas at El Paso. He holds a Ph.D. degree in Computational Science Program. He has years of research experience across projects in image data mining, machine learning, deep learning, and computer simulation for industrial and healthcare applications. In addition, Dr. Rahman has taught various engineering courses in industrial and manufacturing engineering. His research area covers advanced quality technology, AI applications in smart manufacturing, health care applications, computational intelligence/data analytics, and decision support systems.

Selim Molla, University of Texas at El Paso

Selim Molla is a PhD student in the Computational Science Program at the University of Texas at El Paso (UTEP), USA. He received his M.Sc. in Computational Science from UTEP and his B.Sc. in Industrial and Production Engineering from Khulna University of Engineering and Technology (KUET), Bangladesh. His research focuses on computational modeling, simulation, and optimization of complex systems, with applications in manufacturing, industrial, and healthcare domains. He also works on integrating machine learning and deep learning with simulation to support data-driven decision-making and intelligent system design.

Prof. Tzu-liang Bill Tseng, University of Texas at El Paso

Dr. Bill Tseng is a Professor and Chair of Department of Industrial, Manufacturing and Systems Engineering at the UTEP. He is also a Director of Research Institute for Manufacturing & Engineering Systems, the host institute of Texas Manufacturing Assistan

Dr. Richard Y Chiou, Drexel University

Dr. Richard Y. Chiou is a Professor within the Department of Engineering, Leadership, and Society at Drexel University, Philadelphia, USA. His educational background is in manufacturing with an emphasis on mechatronics. In addition to his years of industrial experience, he has taught many different engineering and technology courses at undergraduate and graduate levels. His tremendous research experience in manufacturing includes intelligent manufacturing, eco-manufacturing, Internet based robotics, and clean energy and energy efficiency. He has conducted many experimental studies on virtual reality robotics, ultrasonic robotic welding, Internet-of-Things (IoT) in intelligent manufacturing. He has also developed augmented reality/virtual reality (AR/VR) robotics and clean energy and energy efficiency laboratory recently. He has secured many research and education grants from the National Science Foundation, the US Department of Education, the Society of Manufacturing Engineers Education Foundation, and industries. Techniques developed from his research projects have been extended to the educational activities involving intelligent manufacturing with clean energy and energy efficiency.

Dr. Md Abdul Quddus, North Carolina State University at Raleigh

Dr. Md Abdul Quddus, PhD is an Assistant Professor in the Department of Textile Engineering, Chemistry and Science at NC State University and Associate Faculty in the Operations Research Graduate Program. His research focuses on supply chain and logistics, big data analytics, stochastic programming, geospatial analytics for optimization, and simulation. Furthermore, his specialties also include on the application of AI, deep learning, cloud computing, and operations research techniques to solve large scale supply chain network and risk management problems. Dr. Quddus has over five years of industry experience as a Senior Operations Research Advisor at FedEx Express. His research has been published in journals such as Expert Systems with Applications, Transportation Research Part E, International Journal of Production Economics, and Applied Energy.

A Risk-Free Approach to Validate Drone Flight Paths Using Computer Simulation and Reinforcement Learning

Abstract

Validating autonomous drone flight paths in physical environments poses safety risks and limits hands-on experimentation in engineering education. To address this challenge, this study presents a simulation-based learning platform that integrates computer modeling and reinforcement learning (RL) to support experiential learning in drone navigation and control. A 3D grid-based drone workspace was developed in FlexSim and connected to the Stable Baselines3 Proximal Policy Optimization (PPO) algorithm through a custom Gym-compatible Python interface. Within this virtual environment, students can observe and experiment with an RL-controlled drone learning to reach target locations while avoiding obstacles and boundary collisions. The effectiveness of the learned policies is evaluated using quantitative metrics such as collision rate, path length efficiency, and task success rate, enabling both performance assessment and instructional discussion. The platform has been introduced as a laboratory-style learning module, allowing students to visualize policy learning, trial-and-error decision-making, and fail-safe behavior in a classroom or laboratory setting. The framework demonstrates how simulation-driven RL environments can enhance teaching and learning in robotics, controls, and artificial intelligence courses.

Keywords: Computer Modeling, Reinforcement Learning (RL), Proximal Policy Optimization (PPO), Drone Simulation, Simulation-Based Learning, Engineering Education.

1 Introduction

Simulation has long been a cornerstone of engineering education. It allows students to explore complex systems safely, repeat experiments, and test design decisions without the cost or risk of physical hardware [1]. At the same time, reinforcement learning (RL) has emerged as a practical control and decision-making method for systems that operate under uncertainty [2]. These include robotics, manufacturing automation, and autonomous vehicles [3], [4]. Despite their shared relevance, simulation and RL are often taught separately. Students may learn RL theory through simplified grid worlds, while industrial simulation tools are reserved for deterministic process modeling. This separation creates a gap between conceptual understanding and applied competence.

The need for tighter integration has become more apparent as autonomous systems move into real operational contexts. Drones provide a clear example. They are now used across medicine, agriculture, logistics, construction, media, and emergency response, where fast deployment and adaptive path planning are critical. In such applications, flight paths must respond to obstacles, time constraints, and changing environments [2], [5]. Teaching these ideas purely through equations or static examples limits student intuition. Simulation-based drone environments, combined with RL, allow students to observe how policies evolve, how agents respond to feedback, and how performance metrics change over time [3]. This kind of interaction supports deeper learning and mirrors the challenges students will face in practice [6] – [8].

However, most educational RL platforms rely on abstract environments that do not resemble industrial systems. Although useful for introducing algorithms, they rarely expose students to realistic modeling assumptions, state representations, or operational constraints found in manufacturing and automation. Conversely, commercial simulation software often lacks native support for modern RL workflows and makes it difficult to use as a learning platform for intelligent control. As a result, students struggle to connect RL concepts such as reward design, exploration, and policy convergence to realistic engineering systems [9]. This work addresses that gap by integrating an industrial-grade simulation environment with contemporary RL frameworks through a custom interface. The platform enables students to train and evaluate RL agents inside a three-dimensional simulation that reflects real-world automation and drone navigation scenarios. Rather than treating RL as a black box, students interact directly with the agent–environment loop. They can visualize state transitions, observe agent behavior, and measure performance using meaningful metrics such as path efficiency and task completion time.

The educational motivation behind this work is explicit. The target audience includes upper-level undergraduate and graduate engineering students in manufacturing, robotics, automation, and related fields. The primary learning objectives are threefold. First, students develop an operational understanding of RL concepts by interacting with agents rather than only deriving equations. Second, they visualize how agents perceive and act within a simulated environment, strengthening intuition about dynamics, constraints, and feedback. Third, they connect theoretical RL constructs to industrial-style simulation models that resemble those used in professional practice.

The platform is designed to fit directly into existing engineering coursework, supporting guided laboratory exercises as well as open-ended assignments and capstone-style projects. Students can modify reward functions, state representations, and environment parameters, and then immediately observe how these choices affect agent behavior and performance. From a technical standpoint, this work introduces a modular integration of industrial simulation software with a standard reinforcement learning training pipeline, which enables real-time interaction between learning agents and realistic environments. From an educational standpoint, it shows how such an integration can support active learning, strengthen conceptual understanding, and develop applied skills that connect emerging AI methods with real-world engineering systems.

2 Background and Related Work

2.1 Educational Use of Reinforcement Learning

Recent work has explored the use of reinforcement learning as an instructional tool in engineering and computer science education. Several studies report improved student engagement when RL concepts are introduced through interactive environments rather than static lectures.

Henderson et al. [10] show that commonly used RL benchmarks, while useful for conceptual illustration, often lack realism and can obscure performance variability. More recent educational studies emphasize the need to expose students to constraints that resemble real systems. Garcia and Fernández [11] highlight that many RL formulations abstract away safety constraints and operational limits, and students often struggle to transfer RL knowledge from abstract environments to physical or industrial contexts. Similar observations are reported by Dulac-Arnold et al. [12], who argue that reinforcement learning systems trained in simplified environments often

fail to capture sensing uncertainty, delayed rewards, and complex system dynamics encountered in real-world applications.

Despite these efforts, most instructional platforms still rely on lightweight simulators or highly abstract tasks. As a result, RL is often perceived by students as an algorithmic exercise rather than an engineering control methodology.

2.2 Simulation-Based RL Platforms

Simulation environments play a central role in RL research and development. They provide a controlled setting where learning agents can interact with complex dynamics, receive feedback, and be evaluated systematically. For example, Juliani et al. [13] demonstrate game-engine–based RL environments that enable users to experiment with reward design, state representations, and policy learning within configurable simulations. These environments emphasize flexibility and scalability, allowing users to modify task logic and observation structures while remaining compatible with standard RL training pipelines. Physics-based simulators such as MuJoCo, Gazebo, and Isaac Gym are widely used to train and benchmark learning agents, particularly in robotics and locomotion tasks [14], [15]. These platforms provide accurate dynamics and fast training but are primarily designed for research workflows and expert users. They offer limited support for instructional use, and their abstractions often obscure the relationship between modeling assumptions, learning behavior, and system-level performance.

Several recent studies explore integrating RL with simulation for autonomous navigation and robotics training. Panerati et al. [16], for instance, present a simulation-driven RL framework for aerial robot control that incorporates realistic physics and multi-agent interactions. Similarly, Koch et al. [17] use simulated environments to study learning-based UAV attitude control under disturbances and uncertainty. While these systems demonstrate strong technical performance and realistic dynamics, they are not designed with educational use in mind. Instructional objectives, learner interaction, and curricular integration are outside their scope, limiting their suitability as teaching platforms.

2.3 Industrial Simulation and Educational Gaps

Industrial-grade simulation tools are widely used in manufacturing and automation for system design, validation, and optimization. Recent work in digital twins and cyber-physical systems emphasizes the importance of high-fidelity simulation in supporting realistic system analysis and lifecycle decision-making [18], [19]. However, these environments typically rely on deterministic control logic and are not designed to support learning agents or adaptive decision-making within the simulation loop. Attempts to combine RL with industrial simulation have largely focused on performance optimization rather than education. Studies such as Zhou et al. [20] and Waschneck et al. [21] apply reinforcement learning to production scheduling and control in simulated factory environments, demonstrating improved operational performance. However, these systems are not accessible to students and provide limited transparency into the learning process.

Taken together, existing work highlights a gap between educational RL platforms that prioritize simplicity and industrial simulation tools that prioritize realism. Few systems are designed to

bridge these domains in a way that supports both technical learning and instructional use. This gap motivates the integrated simulation–reinforcement learning platform described in the next section.

3 Methodology

3.1 System Architecture

The platform is a closed loop among FlexSim (simulation and visualization), a custom Gym-compatible Python interface (translation layer), and Stable Baselines3 (reinforcement learning). This setup keeps the simulator visual and realistic while keeping the learning algorithm modular and reproducible. FlexSim runs the 3D GridWorld3D model and maintains the world state at each step: drone position, target location, obstacle occupancy, and boundary conditions. It also provides the on-screen animation that students can watch during training and testing. That visual feedback matters instructionally because learners can connect policy behavior to concrete events (e.g., repeated wall hits, safer route discovery) rather than treating training as a black box.

The Python Gym wrapper exposes the simulator through the standard reset() / step() interface expected by Stable Baselines3. It (1) converts FlexSim's raw state into a consistent observation vector and defines fixed action/observation spaces, and (2) synchronizes step-by-step communication. Each cycle follows a request–response pattern: FlexSim sends an observation, Python returns an action, FlexSim applies the action and advances one discrete step, and then returns the next observation plus event flags (collision, out-of-bounds, target reached). The wrapper returns the Gym tuple (observation, reward, termination, info) and logs step/episode data.

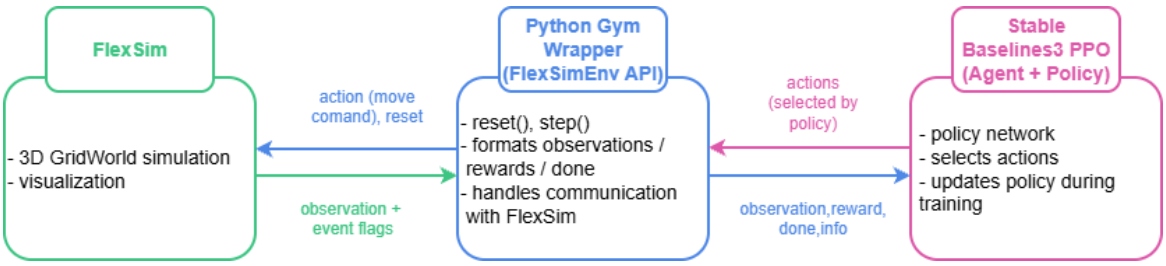

Figure 1. Learning objectives of the three different project phases

To support both training and real-time testing, the FlexSim Reinforcement Learning module was used to implement step-based interaction inside the simulation. FlexSim and Python communicate through a request–response loop: FlexSim sends an observation and requests an action, and Python returns the selected action. In training, Python also computes rewards, checks termination conditions, and logs step- and episode-level metrics. In inference, the same pipeline is used, but actions come from a loaded PPO policy rather than an updating learner. Using one shared pipeline prevents train–test mismatches and keeps evaluation runs consistent and clean.

On the learning side, Stable Baselines3 PPO treats the wrapped FlexSim environment like any other Gym environment. The training script initializes the environment, validates it against Gym API expectations, and then runs PPO rollouts. PPO collects trajectories (state, action, reward, next state) from the environment, updates the policy network, and repeats. Since the simulator is external, reproducibility depends on the wrapper being explicit about seeds, reset behavior, and termination rules. For that reason, the environment reset routine reinitializes the drone, target, and

obstacle configuration in a controlled way before each episode, and the wrapper enforces consistent step timing and return formats. This architecture also supports learning implicitly: students can experiment at the level of observations, actions, and reward design without modifying FlexSim internals, while still seeing behavior through visualization and logged monitoring outputs.

3.2 Simulation Environment Design

The environment is a discrete **7 × 7 × 7 GridWorld3D** implemented in FlexSim, resulting in **343 total states**. The drone (agent) starts from a specified cell and must reach a goal cell while avoiding obstacles and remaining within the workspace boundary**.** Figure 2 illustrates the setup: the agent is visualized as a quadrotor, obstacles appear as red cubes, and the goal region is marked in green. The transparent boundary makes constraints visible, so students can see immediately why a move is valid or not.

Obstacles and boundary constraints: Obstacles are implemented as occupied cells (red cubes). Attempts to enter an occupied cell are treated as invalid or penalized transitions. Boundary violations are also explicitly detected and penalized. Keeping obstacles and boundaries visible supports learning by turning constraint violations into observable events rather than hidden code behavior.

Grid and step dynamics: Each grid cell represents one admissible drone position. The agent moves in discrete steps, transitioning to a neighboring cell at every decision point. This keeps the task simple to inspect while still showing trial-and-error learning and gradual improvement in path quality.

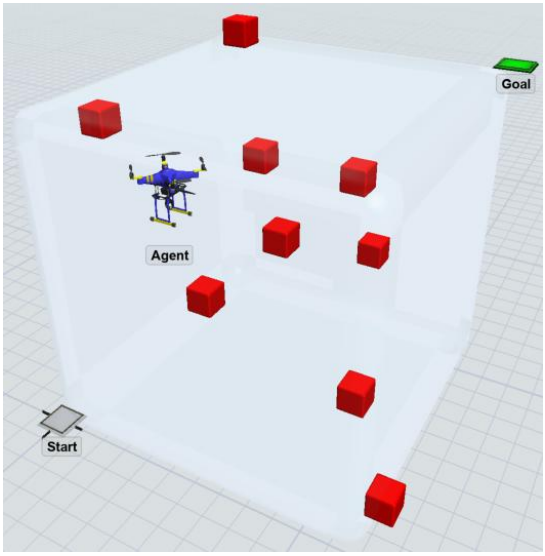

Figure 2. GridWorld3D simulation environment in FlexSim.

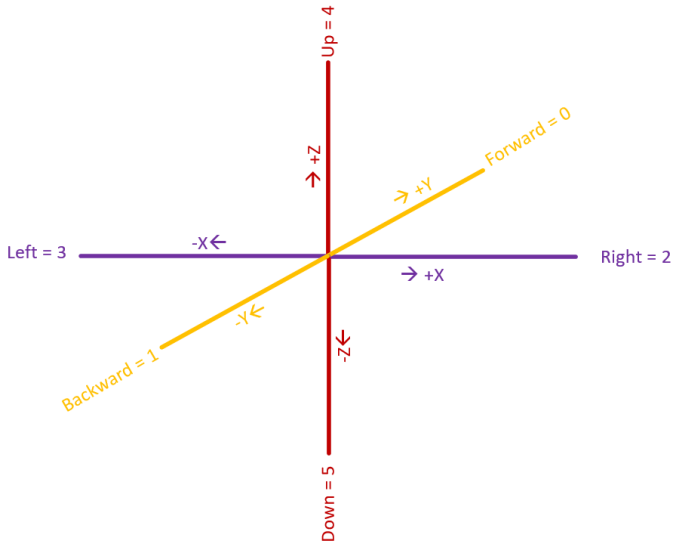

Figure 3. Action definition of GridWorld3D.

Action space: The agent selects one of six discrete actions aligned with the Cartesian axes: Forward (+Y), Backward (−Y), Right (+X), Left (−X), Up (+Z), and Down (−Z). Actions are

encoded as integers: Forward = 0, Backward = 1, Right = 2, Left = 3, Up = 4, Down = 5 (Figure 3). The same mapping is implemented in FlexSim through a ProcessFlow decision block that routes each action to the corresponding motion primitive (Figure 4).

State encoding: The drone's location is encoded as a single integer state index derived from its grid coordinates (x, y, z). As shown in Figure 5, the index is computed as:

$$\boldsymbol{Agent\ State}, s = x + 7y + 49z \qquad (1)$$

This compact encoding supports consistent communication between FlexSim and Python and helps students relate logged trajectories back to grid positions.

Reward and penalties: The reward logic is defined directly in the environment to encourage goal-reaching, safe motion, and short paths:

$$\boldsymbol{Reward}, r_t = \begin{cases} +10, & \text{if } s_{t+1} \in G \text{ (goal)} \\ -5, & \text{if obstacle or boundary violation occurs} \\ -1, & \text{otherwise (free - space step cost)} \end{cases}$$

The step cost discourages wandering, while obstacle and boundary penalties push the policy away from unsafe actions.

Because these per-step executions are implemented as visible blocks in the FlexSim ProcessFlow script (Figure 4), students can adjust reward weights, obstacle logic, or boundary checks without needing to rewrite the RL training code, which supports controlled experiments in a lab setting.

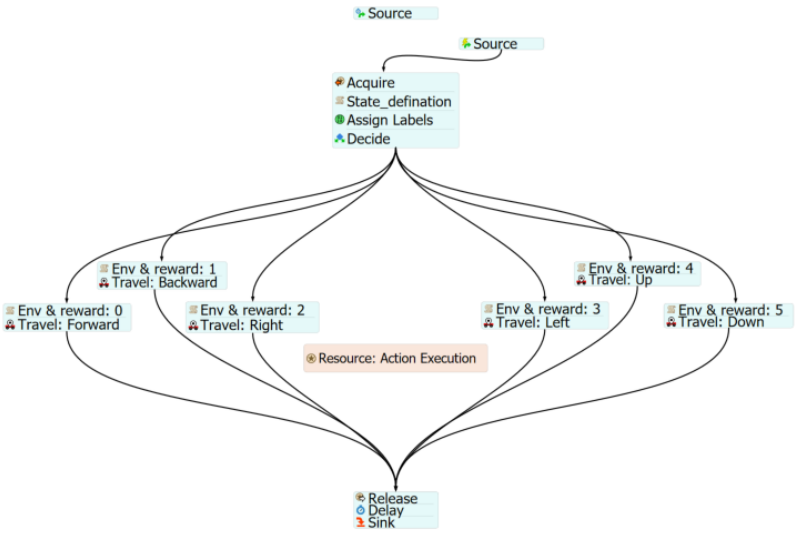

Figure 4. Environment controlling processflow diagram of of GridWorld3D.

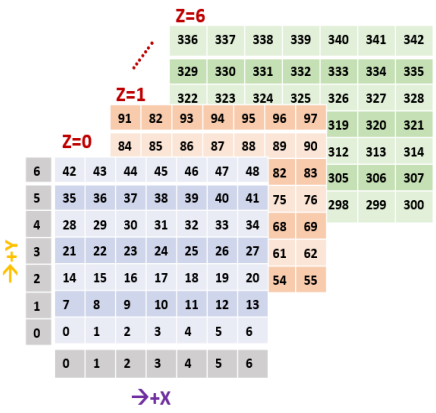

Figure 5: State space of GridWorld3D.

3.3 Reinforcement Learning Formulation

The GridWorld3D is modeled as a discounted MDP:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

At time step t, the agent observes $s_t \in \mathcal{S}$, chooses $a_t \in \mathcal{A}$, and the environment returns $(s_{t+1}, r_t, done_t)$ with

$$s_{t+1} \sim P(\cdot \mid s_t, a_t), \qquad r_t = R(s_t, a_t, s_{t+1}), \qquad a_t \sim \pi_\theta(\cdot \mid s_t)$$

The termination signal $done_t \in \{0,1\}$ is defined as:

$$done_t = 1\{s_{t+1} \in \mathcal{G} \vee t+1 = T_{max}\}$$

The concrete specifications of $\mathcal{S}$, $\mathcal{A}$, obstacle/boundary handling, and the reward table are defined in Section 3.2 (Figures 2 - 5).

The learning signal is the discounted return:

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k}$$

Learning maximizes the expected discounted return:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right]$$

Proximal Policy Optimization and training setup:

Policy learning is performed using Proximal Policy Optimization (PPO) in Stable Baselines3 with an actor–critic setup: a policy $\pi_\theta(a \mid s)$and a value function $V_\phi(s)$. PPO was selected because it trains reliably in practice and fits well with discrete action spaces, where exploration naturally emerges from a stochastic policy.

Training follows the standard rollout-and-update cycle:

1. *Collect rollouts:* interacting the current policy in the Gym-wrapped FlexSim environment.

$$\text{Trajectories} \sim (s_t, a_t, r_t, s_{t+1}, done_t)$$

2. *Compute returns and advantage estimates:* from the collected rollouts using temporal-difference errors:

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \text{ and GAE}(\lambda)\text{: } \hat{A}_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}$$

3. *Update the policy network:* PPO updates θ by maximizing the clipped surrogate objective (to limit overly large policy shifts).

   - Define the probability ratio: $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
   - Then optimize: $L^{\text{clip}}(\theta) = \mathbb{E}_t[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$
   - Optionally, include an entropy term to support exploration: $L^H(\theta) = \mathbb{E}_t[\mathcal{H}(\pi_\theta(\cdot \mid s_t))]$

   So the actor update is: $\max_\theta \left(L^{\text{clip}}(\theta) + c_H L^H(\theta)\right)$.

The rollout buffer is shuffled and split into minibatches; the policy update is applied for multiple epochs per rollout.

4. *Update the value (critic) network:* by minimizing,

$$\min_{\phi} L^{V}(\phi) = \mathbb{E}_t\left[\left(V_\phi(s_t) - \hat{V}_t\right)^2;\right] where, \hat{V}_t = \hat{A}_t + V_\phi(s_t)$$

The critic is updated on the same minibatches (typically in the same training epochs).

5. *Repeat steps 1–4:* until the configured total timesteps are reached.

During training, policy is stochastic so actions are sampled from $a_t \sim \pi_\theta(\cdot|\, s)$ to encourage exploration. During evaluation, the trained policy follows deterministic action selection ($a_t = \underset{\phi}{arg\,max}\ \pi_\theta(a \mid s_t)$) to produce stable trajectories for path monitoring and metric comparison (Section 3.4).

3.4 Path Monitoring, Evaluation Metrics, and Experimental Procedure

Learning performance is assessed by comparing **how** efficiently the agent reaches the goal under a random baseline versus the trained PPO policy. To keep evaluation transparent and easy to reproduce, each episode is treated as a logged trajectory in the 7×7×7 grid, and the same logs are used for analysis and for student-facing feedback in FlexSim.

Path monitoring: For every run, the system records the trajectory

$$\tau = \{(s_t, a_t)\}_{t=0}^{T-1},$$

along with step-level flags for obstacle interactions and boundary violations. These logs make behavior inspectable: students can replay paths, see where invalid moves occur, and connect visible events to reward penalties.

Primary metrics: Results are reported using:

- *Steps-to-goal (path length):* $L = T_{\text{goal}}$, the number of steps taken until the goal is reached (lower is better).
- *Success rate:* $\text{SR} = \frac{1}{N}\sum_{i=1}^{N} \text{Success}_i$, where $\text{Success}_i = 1$if the goal is reached within the episode horizon.
  Obstacle and boundary event counts are also tracked to support debugging and interpretation, even when they are not the main headline metric.

Experimental procedure. A random baseline is generated using the FlexSim Experimenter tool with 1000 replications, where actions are selected uniformly at random. The trained PPO policy is evaluated on the same obstacle configuration using deterministic action selection to produce repeatable trajectories. Both conditions use identical logging and metric computation, enabling a direct comparison of policy quality (reported in Section 4).

4. Results

4.1 Evaluation of the Simulation-based Drone Training Module

This section compares the learned PPO policy against a random baseline in the 7×7×7 GridWorld3D environment. Two outcomes are reported: steps-to-goal (efficiency) and success rate (reliability). A random baseline was generated using the FlexSim Experimenter with 1000 replications. In this condition, the agent selects one of the six actions uniformly at random. Across these replications, the agent required an average of 3865 steps to reach the goal. Under the same obstacle configuration, the trained PPO policy reached the goal in 19 steps. In step-count terms, the trained policy reduces the effort by roughly 211% times. To assess reliability, both policies were also evaluated using an episode horizon of 10,000 steps. The random policy reached the goal within this horizon in 80% of runs, while the trained PPO policy achieved 100% success.

Table 1. Steps-to-goal comparison (7×7×7 GridWorld3D)

Condition	**Evaluation setup**	**Steps to reach goal**	**Success rate (10,000 step horizon)**
Random policy	1000 replications (Experimenter)	3865 (average)	80%
PPO policy	Deterministic evaluation	19	100%

The numbers match what is visible in the simulator. Random behavior produces long, wandering paths with frequent revisits. The PPO policy produces a short, directed trajectory that respects the obstacle layout and boundary constraints. Because trajectories and event flags (obstacle interactions, boundary violations) are logged, the same runs can be replayed and inspected step by step. That makes it easier to explain *why* a policy is efficient, not just that it is.

4.2 Classroom Implementation and Evaluation Plan

The module has been fully implemented and tested to ensure functional reliability, usability, and instructional feasibility; however, it has not yet been deployed in a formal classroom setting. We plan to integrate this module into the Computer Simulation Class during the Fall 2026 semester, where student engagement, conceptual understanding, and learning gains will be systematically evaluated using pre/post assessments and feedback instruments. Empirical learning outcome results will be reported in a subsequent extension of this work, once classroom implementation and data collection are completed.

5. Conclusion

This work introduced a simulation-based learning platform that links a FlexSim GridWorld3D drone environment to Stable Baselines3 PPO through a custom Gym-compatible interface. The core contribution is the integration itself: a reproducible RL training loop paired with a simulator that stays visual, editable, and safe for repeated student experimentation. The environment was built so that key elements, such as step execution, reward logic, and boundary/obstacle checks, are explicit in FlexSim rather than hidden within the RL code. That design choice matters for teaching. Students can inspect behavior as trajectories, adjust environment rules in a controlled way, and immediately see how those changes affect learning.

The evaluation approach is deliberately simple: track trajectories, count steps toward the goal, and report success within a fixed horizon. These metrics are easy to explain and directly connect to

what students observe during playback. In practice, the trained policy showed clear improvements over a random baseline, demonstrating that the platform can produce measurable policy learning while remaining understandable at the classroom level. Future work will extend the study in three directions: (1) evaluate across multiple obstacle layouts and start–goal pairs to test generalization, (2) report additional safety and efficiency summaries from the logged event flags, and (3) formally study learning outcomes when the module is used in coursework, or by observing how students use the monitoring tools and how their understanding of RL concepts changes across lab activities.

References

[1] A. Negahban, "Simulation in engineering education: The transition from physical experimentation to digital immersive simulated environments," *Simulation*, vol. 100, no. 7, pp. 695–708, Jul. 2024, doi: 10.1177/00375497241229757.

[2] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, May 2015, doi: 10.1038/nature14542.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int J Rob Res*, vol. 32, no. 11, pp. 1238–1274, Sep. 2013, doi: 10.1177/0278364913495721.

[4] L. Monostori, "Cyber-physical Production Systems: Roots, Expectations and R&D Challenges," *Procedia CIRP*, vol. 17, pp. 9–13, 2014, doi: 10.1016/j.procir.2014.03.115.

[5] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, vol. 91, pp. 99–131, May 2017, doi: 10.1016/j.paerosci.2017.04.003.

[6] M. Prince, "Does Active Learning Work? A Review of the Research," *Journal of Engineering Education*, vol. 93, no. 3, pp. 223–231, Jul. 2004, doi: 10.1002/j.2168-9830.2004.tb00809.x.

[7] R. M. Felder and R. Brent, "LEARNING BY DOING," 2003. [Online]. Available: http://www.ncsu.edu/felder-public/Columns/FAQs-2.html.

[8] D. A. Kolb, "Experiential Learning: Experience As The Source Of Learning And Development Executive skills of Family Medicine Faculty View project How You Learn Is How You Live View project." [Online]. Available: http://www.learningfromexperience.com/images/uploads/process-of-experiential-learning.pdf!

[9] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of Real-World Reinforcement Learning," Apr. 2019, [Online]. Available: http://arxiv.org/abs/1904.12901

[10] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning That Matters," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018, doi: 10.1609/aaai.v32i1.11694.

[11] J. García and F. Fernández, "A Comprehensive Survey on Safe Reinforcement Learning," 2015.

[12] G. Dulac-Arnold, D. Mankowitz, and T. Hester, “Challenges of Real-World Reinforcement Learning,” Apr. 2019, [Online]. Available: http://arxiv.org/abs/1904.12901

[13] A. Juliani *et al.*, “Unity: A General Platform for Intelligent Agents,” May 2020, [Online]. Available: http://arxiv.org/abs/1809.02627

[14] . IEEE Staff, *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012.

[15] V. Makoviychuk *et al.*, “Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning,” Aug. 2021, [Online]. Available: http://arxiv.org/abs/2108.10470

[16] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, “Learning to Fly -- a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control,” Jul. 2021, [Online]. Available: http://arxiv.org/abs/2103.02142

[17] W. Koch, R. Mancuso, R. West, and A. Bestavros, “Reinforcement learning for UAV attitude control,” *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, Feb. 2019, doi: 10.1145/3301273.

[18] E. H. Glaessgen and D. S. Stargel, “The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles.”

[19] F. Tao, Q. Qi, A. Liu, and A. Kusiak, “Data-driven smart manufacturing,” *J Manuf Syst*, vol. 48, pp. 157–169, Jul. 2018, doi: 10.1016/j.jmsy.2018.01.006.

[20] L. Zhou, L. Zhang, and B. K. P. Horn, “Deep reinforcement learning-based dynamic scheduling in smart manufacturing,” in *Procedia CIRP*, Elsevier B.V., 2020, pp. 383–388. doi: 10.1016/j.procir.2020.05.163.

[21] B. Waschneck *et al.*, “Deep reinforcement learning for semiconductor production scheduling,” in *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, IEEE, Apr. 2018, pp. 301–306. doi: 10.1109/ASMC.2018.8373191.