Extended Abstract

Motivation As drones take on increasingly complex real-world tasks like navigating dense forests or disaster zones, demand for robust autonomous navigation systems has grown. However, prior work focuses on static or structured environments, leaving the problem of generalizing to randomized, obstacle-filled 3D spaces relatively underexplored. Traditional planning methods struggle with adaptability. While reinforcement learning (RL) shows promise, many RL-based UAV studies fail to evaluate generalization across unseen configurations. We address this gap by comparing Soft Actor–Critic (SAC) and Behavior Cloning (BC) in a custom simulated environment featuring dynamic tree placements and randomized start-goal conditions. We aim to evaluate which method better supports robust and adaptable autonomous drone navigation in complex, unpredictable environments.

Method We designed our project using using the drone PyBullet simulator `RL-pybullets-cf` Binschmid (2022), and used it to model our drone agent navigating through our generated 3D environments with static obstacles. As part of our simulations, we model the agent as our drone, define a goal state, and randomly generate trees as obstacles in our environment. We custom designed a reward function that incorporates the following components to determine the action's reward to the drone: The step component to encourage the drone to minimize the number of total steps the drone takes, the collision component to penalize collisions and proximity to trees, the goal component that rewards the drone for reaching the goal and moving towards the goal, and a stability component to encourage the drone to move in a stable manner. We also define two models, a Behavioral Cloning (BC) model that we use as our baseline, and it uses expert trajectories from the A* planner to train a policy $\pi_\theta(a \mid s)$ via supervised learning. Our main model is Soft Actor Critic (SAC), which learns a stochastic policy by maximizing expected return and entropy, and in our implementation, we take the minimum of two critics to reduce overestimation bias.

Implementation We generate a PyBullet scene with an open field (0 trees) or a forest with 20–30 randomly placed trees. For each training episode, we generate a new scene, with random tree sizes, shapes, and locations in the forest scenario. Next, we trained the BC and SAC models. For BC, we generate a fixed dataset of expert trajectories via an A* planner offline, and train a behavioral-cloning policy on these demonstrations for 100 epochs. For SAC, we initialize SAC's actors and twin critics and train the policy for 750 episodes using our reward function. We evaluate the models on both the open field environment and the forest environments, using 1000 rollouts per environment type. We report Success Rate, Average Reward, Collision Rate, and Average Steps for the successful runs.

Results Results are in Table 1. Without trees, BC outperforms SAC with the highest average reward (-30.38). SAC outperforms BC with trees with the highest success rate (45.1%) and less collisions. Average episode length is similar between both models with trees, but different (and greater) without.

Discussion BC outperforms SAC in environments with trees, likely due to SAC heavily relying on obstacle terms in its reward function for strong reward signal. SAC outperforms BC in environments with trees by unifying planning and control in its reward function, unlike BC expert demonstrations, which separate the two between an A* planner and PID controller. This also demonstrates the benefit of penalizing collisions with and proximity to obstacles through reward terms. SAC and BC both perform better with trees, suggesting that SAC relies on the presence of obstacles to avoid sparse rewards, while BC may over fit in simpler environments. It is crucial that the relative weighting of SAC reward terms does not allow the agent to maximize rewards without actually reaching the goal. We refined the reward function to further encourage proximity to and arrival at the goal.

Conclusion and Future Work SAC's shortcomings without trees cannot be resolved by modifying the reward function when training in such environments (perhaps by removing obstacle terms or increasing weights on other terms). This would prevent the training of a general model in environments both with and without trees (which requires the same reward function to be used in all of those environments). Instead, the SAC actor can be initialized to BC's distribution, which has consistent performance in environments with and without trees. This may reduce the effects of overestimation or randomness in initial Q values. Training SAC in a more off policy manner may improve efficiency when environments are complex (so rolling out often is expensive). Our environment's complexity can be extended with conditions like moving obstacles and extreme weather.

Forest or Field: It Drone Matter

Sarah Barragan
Department of Computer Science
Stanford University
sabarrag@stanford.edu

Karan Bhasin
Department of Computer Science
Stanford University
ksbhasin@stanford.edu

Sukrut Oak
Department of Computer Science
Stanford University
sukrut@stanford.edu

Abstract

We aim to contribute to the field of drone navigation research by designing environments with randomly placed obstacles and goals, and by training an agent to optimally navigate such environments. We experiment with both a behavioral cloning model and a soft actor critic model, and compare the performance of the two in environments both with and without trees. We find that SAC falls short of BC's performance in environments without trees due to sparse reward signal, but that is exceeds BC's performance in environments with trees by unifying planning and control in a reward function that also enforces useful penalties for proximity to and collisions with obstacles. A challenge was ensuring that SAC did not find ways to maximize rewards without actually reaching the goal. Future work entails training a general model that works across environments with and without trees, increasing the environment's complexity in order to further challenge the agent, and improving SAC model efficiency through more off policy techniques.

1 Introduction

The use of drones, or unmanned aerial vehicles (UAVs), has risen dramatically in recent years, with applications expanding across a wide range of fields. In agriculture, drones equipped with multispectral and thermal sensors are transforming crop management through disease monitoring and yield estimation, giving farmers actionable insights to optimize resources and improve sustainability Folio3 AgTech (2024). In the retail sector, companies like Walmart, in partnership with Alphabet's Wing, are rolling out drone delivery services in select U.S. cities, offering customers the ability to receive groceries and household items in under 30 minutes Marshall and Dave ([n. d.]). Yet the role of drones extends beyond consumer convenience. In emergency response, drones are increasingly vital for search and rescue missions in disaster zones, where their ability to access hard-to-reach areas and provide real-time data can save lives Ghosh (2025).

As drones take on more dynamic and complex tasks, the need for robust autonomous navigation systems that operate without human intervention becomes critical. Traditional systems that rely on static maps or human control fall short in unpredictable environments. For drones to perform effectively in real-world settings, they must be able to perceive their surroundings and make real-time decisions Aburaya et al. (2024). Reinforcement learning (RL) offers a compelling solution. Rather than being dependent on pre-labeled data, RL agents learn through trial and error by using reward and penalty signals to guide their behavior Liu et al. (2024). This makes RL particularly suitable for navigation tasks in uncertain, obstacle-rich environments where flexibility and adaptability are key.

Among RL methods, Soft Actor–Critic (SAC) has emerged as especially effective. Liu et al. (2024) showed that SAC enabled robots to navigate dynamic environments using LiDAR, achieving faster training and higher success rates. SAC has also shown strong performance in vision-based drone navigation, where adapting to new surroundings is crucial Aburaya et al. (2024). Compared to supervised approaches like Behavioral Cloning (BC), which rely on expert demonstrations and often fail to generalize to novel environments, SAC learns directly from experience and adjusts its strategy as conditions change Tedrake (2024).

Our project investigates the use of Soft Actor–Critic for enabling drones to autonomously navigate randomized, obstacle-filled 3D environments. We use a custom PyBullet simulation populated with randomly placed trees, requiring the drone to reach a goal from a random start without prior environmental knowledge. We evaluate SAC's performance against a Behavior Cloning baseline to assess which method generalizes better to unseen obstacle configurations.

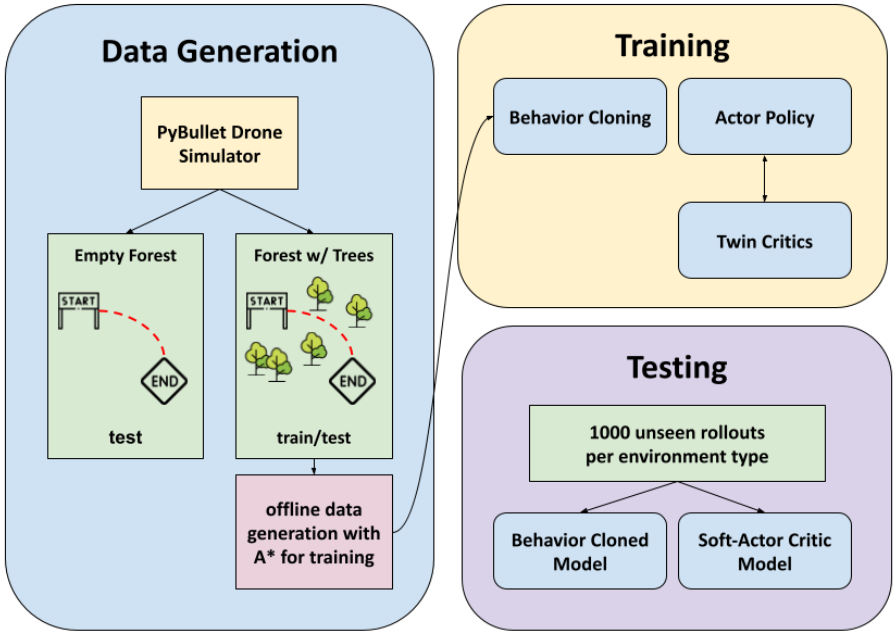

Figure 1: Simulation Workflow Diagram

2 Related Work

Reinforcement learning (RL) has become an increasingly popular approach for enabling autonomous aerial navigation, especially in environments where traditional planning and control methods may struggle due to unpredictability or partial observability. A broad range of research has explored how RL can help UAVs avoid obstacles, maintain flight stability, and reach target locations. However, relatively few studies tackle the more difficult task of navigating randomized, obstacle-filled 3D environments while ensuring robust generalization across unseen configurations.

Several prior works explore RL algorithms such as DQN and PPO. For example, Kalidas et al. (2023) evaluated DQN, PPO, and SAC for UAV navigation across complex environments including woodland scenes and dynamic objects. Their study found that DQN, limited by its discrete action space, produced abrupt, inefficient flight patterns in cluttered spaces, while PPO underperformed in large, vision-based environments due to its on-policy inefficiency and poor sample reuse. Although PPO was more stable than DQN, its learning performance degraded as environment complexity increased. These findings are echoed by Chikhaoui et al. (2022) who introduced a PPO-based UAV framework that achieved up to 90% success in 3D obstacle navigation tasks. However, as the number of obstacles increased so did the performance of the agent. Additionally, their model relied on a simplified discrete action space and was prone to decision paralysis in cluttered scenes, where the

UAV would occasionally stall mid-flight, highlighting PPO's limitations in generating responsive and fine-grained control policies for densely populated 3D environments.

By contrast, Soft Actor–Critic (SAC) has emerged as a leading choice for continuous control tasks due to its sample efficiency, stability and effectiveness in continuous control tasks. Kalidas et al. (2023) found that SAC outperformed both DQN and PPO in navigating visually rich environments with complex layouts, though even SAC was limited by the use of fixed goal positions and static obstacle arrangements, reducing its applicability to dynamic or unpredictable missions.

Further demonstrating SAC's strengths, Liu et al. (2024) applied it to LiDAR-based ground robot navigation in cluttered environments, showing faster convergence, higher success rates, and more optimal trajectories than baseline methods. Their agent handled narrow passages and dynamic scenes effectively, underscoring SAC's suitability for continuous control in complex settings. However, the study focused on 2D ground robots with fixed obstacle layouts, lacking the challenges of 3D aerial navigation such as altitude control and dynamic obstacles. Thus, while promising, their findings do not fully address generalization in randomized, 3D UAV environments.

To improve SAC's adaptability to such aerial tasks, Hwang et al. (2023) proposed Stepwise SAC (SeSAC). Designed for disaster scenario UAV simulations, this method trains an agent by gradually exposing it to more difficult and complex environments. As SeSAC gradually increased environment difficulty during training it achieved high success rates in reaching goals while avoiding obstacles. Nonetheless, SeSAC still relied on structured, pre-designed environments and was not evaluated in settings with randomized obstacles or goal distributions, leaving its ability to generalize across unseen configurations without testing.

In contrast to these RL-based methods, Behavior Cloning (BC), a supervised imitation learning approach, is also widely used for UAV control when expert trajectory data is available. However, BC's reliance on demonstration data causes performance to degrade in novel settings. For instance, Bansal et al. (2024) introduced RaCIL, a hybrid of ray-tracing and BC, and found that while BC could imitate expert flights in known environments, it consistently failed in unseen scenarios, requiring reinforcement learning enhancements like GAIL or PPO for reliable performance. Similarly, Wan et al. (2023) showed that even DAgger-based training, which augments BC with corrective feedback, could not ensure robustness in dynamic or visually complex tasks due to its dependence on simulated data.

Given these limitations in both SAC and BC applications to UAV navigation, our project seeks to address this gap by evaluating the performance of Soft Actor–Critic and Behavior Cloning in a more challenging and realistic setting. Specifically, we investigate how well each method enables a quadrotor to navigate randomized, obstacle-filled 3D environments using a custom PyBullet simulation with dynamically placed tree obstacles. Unlike prior work that focuses on static or simplified environments, our approach emphasizes generalization to unseen obstacle configurations and evaluates target position success from randomized start and goal positions. By directly comparing SAC and BC under identical conditions, our project aims to better understand their respective strengths and limitations in the context of robust, real-world UAV navigation.

NOTE: In addition to these methodological gaps, it is also important to consider the accessibility and reproducibility challenges present in prior work. High-fidelity simulators like AirSim Shah et al. (2017) have enabled sophisticated SAC-based aerial learning in realistic environments, but they come with hardware requirements that often necessitate GPU-enabled systems and complex build configurations. During our own literature and codebase review, we found that many existing SAC implementations for UAVs were either tied to such platforms (rendering them inaccessible on our available machines), overly reliant on static environments, or lacked sufficient open-source documentation for reproducibility.

3 Method

To accomplish the tasks in our problem statement, we forked the drone PyBullet simulator `RL-pybullets-cf` Binschmid (2022), and used it to model our drone agent navigating through our generated 3D environments with static obstacles. As part of our work, we have the following several components to our project.

3.1 Components of our Simulation

Drone (Agent): We model the agent as a quadcopter drone and represent the state using the drone's position, orientation, linear velocity, and angular velocity. The PyBullet simulation software enables us to provide controls to the drone as outputs of our learned reinforcement learning policies. The drone's environment is also represented in the drone's state, including the positions and sizes of the obstacles.

Goal State and Waypoints: The objective of our drone is to fly along a predefined trajectory in 3D space. We generate a sequence of waypoints along the trajectory, and at every timestep, the current goal is the next waypoint on the trajectory. The episode is successful if the drone reaches the final waypoint within a small radius $\epsilon = 3.0$.

Trees (Obstacles): To simulate the obstacles, we generate various numbers of trees to represent different types of environments, as shown in Figure 1. The first set of environments are generated without trees (and thus with no obstacles), while the final set of environments are generated with 20-30 trees randomly placed throughout the space. Each of the trees are also randomly generated to have different sizes, with some trees having shorter trunks and small leaf components and other trees having taller trunks with larger leaf components.

3.2 Examples of Environmental Setups with Random Configuration of Trees

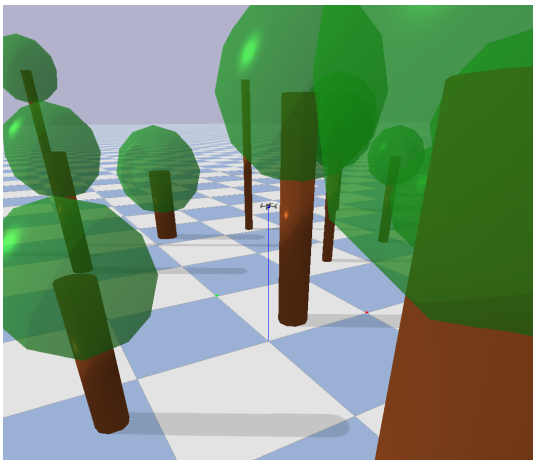

Figure 2: Example 1 of Environmental Setups with Random Configuration of Trees

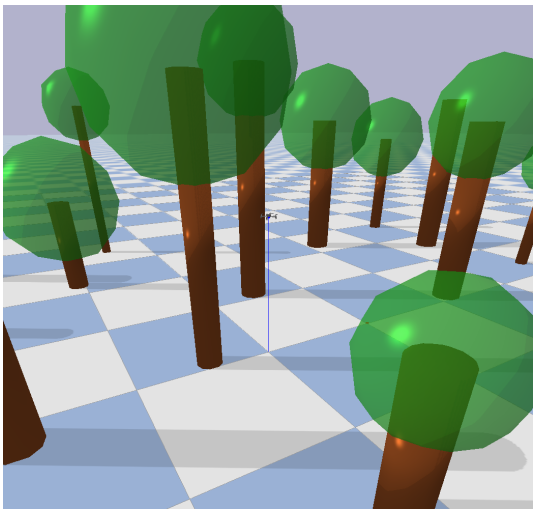

Figure 3: Example 2 of Environmental Setups with Random Configuration of Trees

3.3 Reward Function Components

Our reward function incorporates the following components to determine the action's reward to the drone:

Step Component is a a penalty of 0.5 for each additional step the drone takes, and this is meant to encourage the drone to take action and avoid pausing or not moving. We found that by adding this reward, the drone reached the goal state faster and often prevented the drone from staying in place.

Collision Component is a penalty of 100 for colliding with a tree, and a smaller penalty of 1 when flying too close to a tree. We designed these penalties to heavily discourage the tree from colliding into trees, and the smaller penalty helps guide the drones to stay away from the general vicinity of the trees.

Goal Component consists of a reward of 20 for reaching the goal, and a penalty proportional to the drone's distance from the goal to encourage accuracy. This second component discourages the drone from moving away from its current goal.

Stability Component is a penalty of 10 if the drone tilts excessively. This controls related reward encourages the drone to remain steady and fly with stability.

3.4 Models

Behavioral Cloning (BC): Our first model, which we use as our baseline, is a Behavioral Cloning (BC) Model that uses expert trajectories from the A* planner to train a policy $\pi_\theta(a \mid s)$ via supervised learning. The loss minimizes the negative log-likelihood of expert actions as follows:

$$\mathcal{L}_{\text{BC}} = \mathbb{E}_{(s,a^*)\sim\mathcal{D}} \left[-\log \pi_\theta(a^* \mid s) \right]$$

Soft Actor Critic (SAC): As mentioned previously, our main model that we test and develop for our environments is a Soft Actor Critic (SAC) model, which learns a stochastic policy by maximizing expected return and entropy. The actor maximizes:

$$J(\pi) = \mathbb{E}_{(s,a)\sim\mathcal{B}} \left[Q(s,a) - \alpha \log \pi(a \mid s) \right]$$

The critic minimizes the Bellman error using:

$$y = r + \gamma \mathbb{E}_{a'\sim\pi} \left[Q(s',a') - \alpha \log \pi(a' \mid s') \right]$$

We take the minimum of two critics to reduce overestimation bias.

4 Experimental Setup

An overview of our experimental setup is illustrated in Figure 1. Section 4.1 describes the model training, and section 4.2 describes model evaluations.

4.1 Model Training

1. **Environment Generation:** The first step is to generate a PyBullet scene with an open field (0 trees) or a forest with 20–30 randomly placed trees. For each training episode, we generate a new PyBullet scene, with random sizes, shapes, and locations for the trees in the forest scenario.
2. **Expert Demonstrations and BC Pre-training:** We generate a fixed dataset of expert trajectories via an A* planner offline, and train a behavioral-cloning policy on these demonstrations for 100 epochs.
3. **SAC Fine-tuning:** We initialize SAC's actors and twin critics and train the policy for 750 episodes using our defined reward function. The episodes each terminate on success, after a collision, or after 2400 steps.

4.2 Model Evaluations

1. **Testing:** Both BC and SAC are evaluated over 1000 rollouts per environment type. We report the Success Rate, Average Reward, Collision Rate, and Average Steps for the successful runs.

5 Results

During testing, we rolled out the policies for 1,000 episodes (results displayed in Table 1). Success rate measures the percentage of episodes in which the drone reaches the final goal within ϵ distance, collision rate measures the percentage of episodes in which the drone collides with an obstacle, and average steps is the average length of an episode. We trained and tested both models in environments without trees, and in environments with a random number of trees between 20 and 30, as shown in Figure 4.

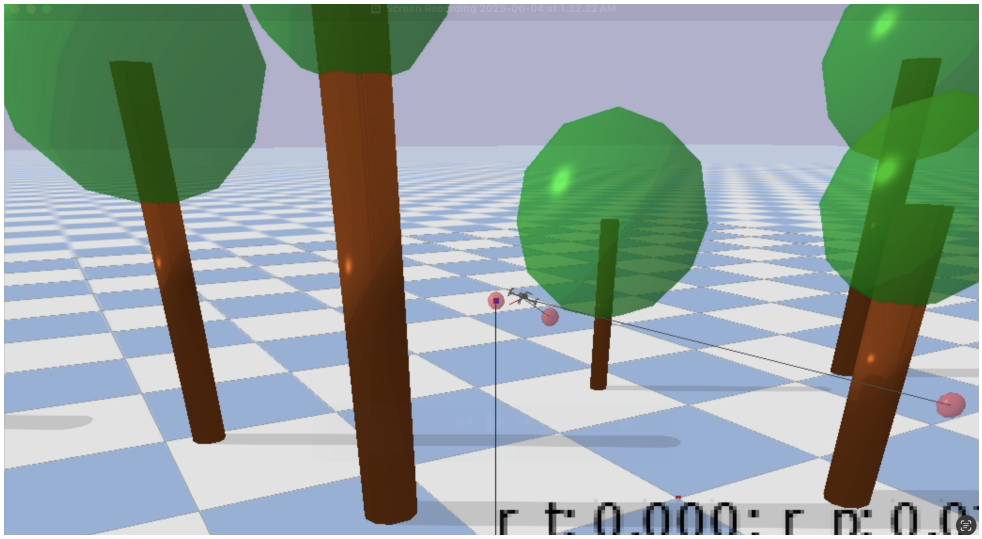

Figure 4: Example Drone Following SAC Policy

Model	Number of Trees	Success Rate	Average Reward	Collision Rate	Average Steps
BC	0	0.39	-30.38	0.0	49.2
SAC	0	0.171	-169.01	0.0	221.3
BC	20-30	0.434	-46.73	0.237	18.3
SAC	20-30	0.451	-38.80	0.168	12.3

Table 1: Test performance of BC and SAC models.

5.1 Qualitative Analysis

5.1.1 Behavioral Cloning (BC)

During training, behavioral cloning, with the A* planner's expert trajectories, worked well to avoid the trees since the A* provided the drone with the best path to the goal. This performance was reflected in the evaluation simulations, especially since the drone was avoiding the trees, moving in the general direction of the target, and visually performed well on the empty field environments as well.

5.1.2 Soft Actor Critic (SAC)

During training, the drone made many mistakes at first, like going in the wrong direction and colliding with the trees. But as the model learned to avoid these behaviors, the drone slowly began navigating around the trees and moving in the correct direction, even in environments with a lot of trees. Unfortunately, this learning wasn't reflected in the open field environments, which might have been because there weren't any trees in the agent's state, so the drone often stayed still and didn't move towards the target, likely caused by some form of reward hacking.

5.2 Quantitative Evaluation

Success Rate: The highest success rate (45.1%) is achieved by SAC in environments that have trees, with BC succeeding only slightly less frequently in such environments. Without trees, the performance of the two models is significantly more different, with BC succeeding 2.28 times as often as SAC.
Average Reward: The highest average reward (-30.38) is achieved by BC without trees, with SAC having a significantly lower average reward of -169.01 in such environments. In environments with trees, the average reward is much more similar between the two models.
Collision Rate: With no trees, the collision rate is 0 for both models since collisions are impossible. With trees, SAC's collision rate is significantly lower than that of BC. Specifically, the BC model collides 1.41 times as often as the SAC model.
Average Steps: The average episode length is similar between both models with trees (12.3 for SAC and 18.3 for BC), but very different without. BC takes 2.69 as many steps without trees compared to with, while SAC takes 17.99 as many.

6 Discussion

- **With no trees, BC outperforms SAC due to weak SAC reward signal. BC's performance is more consistent than SAC's across environments with and without trees.**
  Without trees, BC achieves a notably greater success rate (39.1% compared to SAC's 17.1%) and average reward (-30.38 compared to SAC's -169.01).
  This is likely because the SAC reward function relies on the presence of obstacles in order to provide a strong reward signal, since collisions with and proximity to trees are penalized significantly. Without trees, the rewards are more sparse, and potentially more similar across many different states, making it more difficult for the drone to determine what action to take from a given state. The effects of randomness and overestimation in the initial Q values may be exasperated by the rarity of frequent or extreme rewards. This hypothesis is reinforced by high number of average steps taken by the SAC agent (221.3 compared to BC's 49.2). This behavior potentially reflects the SAC model's struggle to confidently determine the optimal actions to take. The high step count is likely a significant contributor to the low average reward, as each additional step results in an additional penalty.
  Unlike SAC, the BC model does not experience such significant differences in performance between the environments with trees and without. In fact, performance is remarkably consistent (for with trees and without, respectively, the success rates are 43.4% and 39%, the average rewards are -46.73% and -30.38%, and the average steps are 18.3 and 49.2). BC accomplishes this by not relying on a reward function that depends heavily on interactions with obstacles.
- **SAC outperforms BC with trees by accounting for drone control factors and by penalizing collisions with trees. This highlights the benefits of unifying planning and control in a single reward function.**
  With trees, SAC achieves a much lower collision rate (16.8% relative to BC's 23.7%), as well as a greater average reward (-38.80 compared to BC's -46.73) and a greater success rate (45.1% compared to BC's 43.4%).
  This is likely in part because the BC expert demonstrations are generated using two separate techniques: an A* planner that is responsible for planning the trajectory that avoids the obstacles, and a PID controller that is responsible for determining the drone control factors necessary to execute that trajectory. In contrast, the SAC reward function accounts for both trajectory planning factors (proximity to and collisions with obstacles, as well as awarding arrival at the goal and penalizing distance from it), in addition to drone control factors such as tilt and velocity. In doing so, the SAC model ensures that the planned trajectory not only optimizes for efficiently reaching the goal while avoiding obstacles, but also that the planned trajectory is one that can be practically taken by a drone given its control constraints. The fact that a reward function can be an effective way of unifying planning and control suggests that reinforcement learning techniques may be useful in other drone navigation settings, as well as for other kinds of vehicles and robots.
  The BC expert demonstrations, by separating planning and control, fall short of meeting SAC's performance. This highlights a potential shortcoming of the BC planning and control techniques that reinforcement learning techniques like SAC can help to resolve. When

visualizing the BC trajectories, we noticed that the drone would often stay still or fall to the ground. This suggests that the planned A* trajectories did not effectively account for drone control factors. For example, it may be possible, given some environment, for A* to plan an efficient, short, and fast trajectory that avoids all obstacles. However, this trajectory may be unrealistic for a drone to actually practically execute due to its control limitations, resulting in the undesirable behaviors that we observed.
The significantly lower collision rate of SAC compared to BC also helps demonstrate the effectiveness of the obstacle collision and proximity penalties in the SAC reward function. This disparity also suggests that the use of penalties in reinforcement learning techniques can be more effective that the avoidance techniques used in other methods like A*.

- **Both models perform better with trees than without (achieving a higher success rate and average reward), while also taking less steps on average, suggesting more efficient progress toward the goal.**
  This result is expected for SAC, given that in environments with no trees, the reward signal is sparse, resulting in the actor taking many steps that do not result in effective progress in the direction of the goal. For behavioral cloning, this result is more unexpected. Given that the BC model simply emulates experts demonstrations, the quality of the expert demonstrations is the primary determinant of the model's success. Thus, this result suggests that the expert's performance is consistent across environments with and without trees, and that expert performance does not worsen as a result of obstacles being introduced.
  The fact that both models achieve better performance while taking less steps in the environments with trees suggests that, potentially, the added complexity of the environment (through obstacles) is somehow beneficial for encouraging the models to more frequently and efficiently actually arrive at the goal. For SAC, this is likely due to the stronger reward signal, which encourages the model to more decisively take actions from states and make progress in an intended direction, while taking the minimal number of steps needed to do so. For BC, this may be because the neural network over fits to the simpler expert behaviors exhibited in the environments with no trees. When trees are added, the expert may exhibit more complex patterns in its actions and trajectories that the neural network can learn without over fitting. It turn, it also generalizes better during testing, achieving its objective more often and while reducing the number of unnecessary steps.
- **SAC models may attempt to optimize for achieving high rewards without actually accomplishing the task.**
  Early in our experiments, the SAC model achieved a high average reward but a low success rate. We hypothesize that this is because the model learned ways to maximize rewards without actually reaching the objective. We resolved this by adjusting our reward function to further penalize distance from the goal and to further award arrival at the goal, increasingly the effects of these terms relative to the other terms in the reward function. We found these modifications to be quite effective.
  The need for, and benefits of, these adjustments underscores the importance of being wary when applying reinforcement learning techniques, as it is crucial that the reward function accurately reflects the relative importance of the different environmental factors (like obstacles), agent characteristics (like tilt), and task objectives (like reaching the goal). Otherwise, there is a risk that the model seemingly performs well due to its high rewards, but is actually not exhibiting behavior that is considered optimal in practice.

7 Conclusion and Future Work

We hope that our work is helpful for understanding the benefits of SAC over BC for encouraging obstacle avoidance and for unifying planning and control, and the reasons why these models differ in performance. However, it is also important to take note of SAC's shortcomings in the environments without trees, which likely result from reward sparsity. We also hope that our environment can be extended in the future to further challenge the drone, and that our models can be trained more efficiently.

- **Before SAC training, initializing the actor to the trained BC model may help in developing a general model that can be deployed in environments both with and without trees, rather than in only one of these kinds of environments.**

To improve the performance of SAC in the environment with no trees, an intuitive solution is to remove the reward function terms that are related to obstacle collision and proximity, and to increase the weight on other terms such as those related to control and distance to the goal. Doing this may help to increase the density of rewards and in turn allow the actor to more easily determine what action is optimal from a given state. This may also help to reduce negative effects from overestimation and randomness in the initial Q values (for example, if the actor takes an action due to it having a high Q value that does not accurately reflect how useful that action actually is).
However, this solution has a shortcoming, which is that it requires changing the weights of the terms in the reward function depending on the number of trees. This is undesirable because ultimately, it is most useful to develop a general model that can perform well in a variety of environments with vastly differing numbers of trees. Developing such a model requires training in such a variety of environments, and as a result, using the same reward function across those environments.
An alternative solution to the challenge of low SAC reward signal without trees (one that ensures that SAC can still generalize) is to improve how the SAC model is initialized. This can be accomplished by initializing the actor to the trained BC model, and also potentially by assigning a higher Q value to taking action a from state s, relative to the other actions, if the probability of a given s, according to the trained BC model's distribution, is higher relative to the other actions. Given that the BC model has consistent performance both in environments with trees and without, this approach is likely to be effective in reducing the effects of inaccurate initial Q values and in allowing the actor to more easily distinguish between the values of the different actions from a given state.

- **The drone can be trained to accomplish more challenging tasks by extending the environment, perhaps with weather conditions or moving obstacles.**
  One contribution of our work is that our environment can be extended to further challenge the drone and to reflect real world scenarios. For example, drones may often have to navigate through weather conditions like snow or wind, or account for moving obstacles in the air such as other vehicles or birds. Our environment can be modified to include such environmental factors. The reward function must be updated to include any new relevant terms, including penalties, and then the SAC model can be trained and tested in the modified environment. Doing this may be helpful in order to understand how a drone can best respond to the variety of complex and dynamic real world conditions that it is bound to encounter. It may also be helpful to test in environments of different sizes, with different tree densities (for example, a smaller environment with more trees) in order to ensure that the model can generalize.

- **SAC model efficiency could be increased by training in a more off policy manner.**
  The aforementioned environmental modifications may result in a more complex environment, and, as a result, roll outs that are more computationally heavy. Currently, we frequently roll out our SAC policy in order to collect the most relevant and updated trajectories that are most reflective of the current state of our model. We then use these recent trajectories to update the actor and the critics. Doing this is likely quite important for our model's performance. However, this process of rolling out so often, in practice, may be very inefficient and expensive, given that many drone simulation environments may be significantly more complex than the ones that we experimented with. For example, such environments may have many more dynamic environmental factors that interact with each other and with the drone in various different and complicated ways.
  Thus, it could be helpful to experiment with a more off policy version of our SAC model that rolls out the policy less often. Instead, such a training process could entail sampling from a replay buffer that contains older rolled out trajectories. It could then be helpful to weight the relative effects of these sampled trajectories in the update process based on how recent they are and based on how much the model has changed since they were collected.
  Thus, an ultimate future goal could be to efficiently train a general model that can be deployed across environments containing a variety of complex obstacles and other environmental factors.

8 Team Contributions

- **Sarah Barragan** worked on implementing the BC model and generating the environment. We all equally worked on writing up the results, poster, and final paper.
- **Karan Bhasin** worked on implementing the SAC model and running the evaluations. We all equally worked on writing up the results, poster, and final paper.
- **Sukrut Oak** worked on implementing the SAC model and running the evaluations. We all equally worked on writing up the results, poster, and final paper.

Changes from Proposal We updated the contributions based on the work that was needed to set up the environment that we ended up using and based on the time needed to implement our models.

References

A. Aburaya, H. Selamat, and M. T. Muslim. 2024. Review of Vision-Based Reinforcement Learning for Drone Navigation. *International Journal of Intelligent Robotics and Applications* 8, 4 (2024), 974–992. https://doi.org/10.1007/s41315-024-00356-9

Harsh Bansal, Vyom Goyal, Bhaskar Joshi, Akhil Gupta, and Harikumar Kandath. 2024. RaCIL: Ray Tracing based Multi-UAV Obstacle Avoidance through Composite Imitation Learning. In *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*. 2188–2193. https://doi.org/10.1109/CASE59546.2024.10711835

Daniel Binschmid. 2022. RL-pybullets-cf: Reinforcement Learning Environments for Crazyflie in PyBullet. https://github.com/danielbinschmid/RL-pybullets-cf. GitHub repository, commit accessed 4 Jun 2025.

Khalil Chikhaoui, Hakim Ghazzai, and Yehia Massoud. 2022. PPO-based Reinforcement Learning for UAV Navigation in Urban Environments. In *2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 1–4. https://doi.org/10.1109/MWSCAS54063.2022.9859287

Folio3 AgTech. 2024. The Role of Drones in Farming in 2024. Online blog post. https://agtech.folio3.com/blogs/role-of-drones-in-farming-in-2024/ Accessed June 9, 2025.

Baisali Ghosh. 2025. Drones for Search and Rescue Operations. Blog post on FlytBase. https://www.flytbase.com/blog/drones-for-search-rescue Accessed June 9, 2025.

Ha Jun Hwang, Jaeyeon Jang, Jongkwan Choi, Jung Ho Bae, Sung Ho Kim, and Chang Ouk Kim. 2023. Stepwise Soft Actor–Critic for UAV Autonomous Flight Control. *Drones* 7, 9 (2023), 549. https://doi.org/10.3390/drones7090549

Amudhini P. Kalidas, Christy Jackson Joshua, Abdul Quadir Md, Shakila Basheer, Senthilkumar Mohan, and Sapiah Sakri. 2023. Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles. *Drones* 7, 4 (2023), 245. https://doi.org/10.3390/drones7040245

Y. Liu, C. Wang, C. Zhao, H. Wu, and Y. Wei. 2024. A Soft Actor–Critic Deep Reinforcement-Learning-Based Robot Navigation Method Using LiDAR. *Remote Sensing* 16, 12 (2024), 2072. https://doi.org/10.3390/rs16122072

Aarian Marshall and Paresh Dave. [n. d.]. Walmart Goes Big With Drone Delivery Expansion. *Wired* ([n. d.]). https://www.wired.com/story/walmart-wing-expand-drone-delivery/ Accessed June 6, 2025.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2017. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. *arXiv preprint arXiv:1705.05065* (2017). https://doi.org/10.48550/arXiv.1705.05065 v2 revised July 18, 2017.

Russ Tedrake. 2024. Chapter 21 – Imitation Learning. Lecture notes in *Underactuated Robotics*. https://underactuated.mit.edu/imitation.html Available online at underactuated.mit.edu; accessed June 9, 2025.

Yu Wan, Jun Tang, and Zipeng Zhao. 2023. Imitation Learning of Complex Behaviors for Multiple Drones with Limited Vision. *Drones* 7, 12 (2023), 704. https://doi.org/10.3390/drones7120704