Forest Navigation UAV

Final Report

Author: Thodoris Evangelakos
Contact: tevangelakos@tuc.gr
Course: Autonomous Agents - INF412
Date: 27-02-2026

Abstract

With this project we sought to tackle a very common problem for Unmanned Aerial Vehicles (henceforth UAVs): operating in forested areas. Said areas are particularly challenging for UAVs for a number of reasons, such as the many obstacles (trees, branches etc) and the difficulty of wireless communication between the UAV and the pilot. To address this, we used machine learning, taking advantage of more or less known patterns seen in forests to train a policy that can safely and quickly navigate through them. Specifically we trained a Soft Actor-Critic (SAC) agent in a custom fast in-memory simulator (`fastsim`, inspired from this) with a safety shield to prevent collisions, and then ran demo rollouts in Gazebo+ROS2 using generated forest worlds, in worlds twice as dense as the most dense real life examples. We achieved a success rate of 85% in reaching the goal without collisions, with a median time to goal of 12.3s and a shield intervention rate of 15%. While the results are promising, and this project has come a long way since its first iterations, there is still much work to be done to achieve reliable real-world performance, which will be discussed later.

---

1. Problem and goals

1.1 Problem statement

- The task is to enable a UAV to autonomously navigate through a cluttered forest environment and reach a specific goal location as quickly as possible while avoiding collisions.
- The main challenge is balancing the need to reach the goal quickly with the need to avoid collisions, which can be catastrophic for UAVs, especially at high speeds. The environment is also highly unstructured and dynamic, with many obstacles that can occlude the UAV's sensors and create narrow passages that require precise control. Additionally, the UAV must operate under partial observability, relying on limited sensor data to make decisions in real time.
- Failure in this context means that the UAV collides with an obstacle (catastrophic) or fails to reach the goal within a reasonable time frame (suboptimal). The primary failure mode is collision, which can result

in damage to the UAV and its surroundings. Secondary failure modes include getting stuck in local minima (e.g., circling around an obstacle) or taking excessively long paths that are inefficient and may lead to battery depletion.

1.2 Requirements and constraints

- Safety requirements: The UAV must avoid collisions with obstacles at all cost, and later on avoid colliding with other members of the swarm (future work). This is a hard constraint that must be satisfied at all times, and is the primary focus of our safety shield design.
- Performance requirements: The UAV should reach the goal location as quickly as possible, ideally within a time frame that is competitive with human pilots. This is a soft constraint that we optimize for in our reward design, but it is secondary to safety.
- Compute and deployment constraints: The UAV must be able to run the policy in real time on onboard compute, which limits the complexity of the model and the amount of computation that can be done at each step. This is a constraint that we take into account when designing our policy architecture and training setup, and is one of the reasons we chose SAC for its sample efficiency and stability.

1.3 Success criteria

- Success = reaching goal within 30s without collision
- Collision = any contact with an obstacle (detected by simulator or shield)
- Time to goal = time from episode start to reaching goal (only for successful episodes)
- Minimum clearance = minimum distance to any obstacle during the episode
- Shield intervention rate = percentage of steps where the safety shield intervened to modify the action

---

2. Approach overview

2.1 High-level system architecture

The system follows a closed-loop architecture: Sensors -> Observation -> Policy -> Safety shield -> Command -> Simulator/Robot.

Figure 1: End-to-end navigation and safety pipeline.

2.2 Why this approach

- We chose a learned policy because it can adapt to the complex nature of the forest environment and potentially pick up on patterns that are not explicitly programmed. A learned policy can also be more computationally

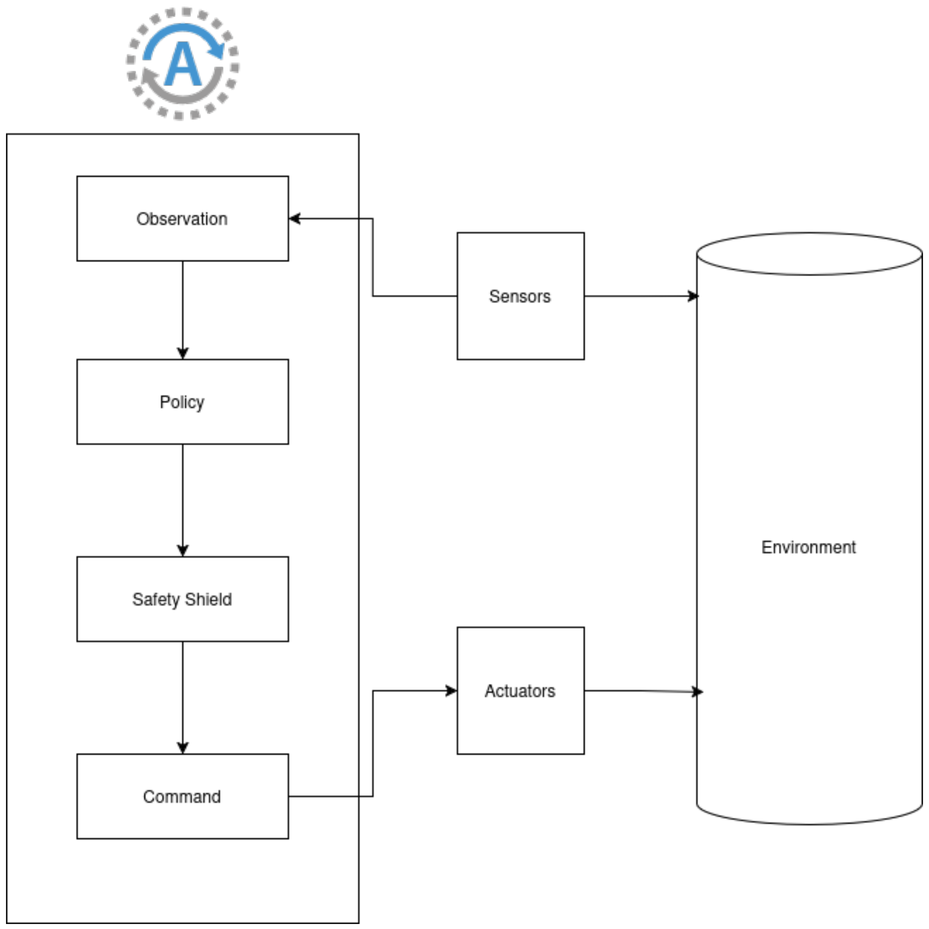


Figure 1: Figure 1 - Pipeline overview

efficient at runtime compared to a traditional planning approach, which is important given our real-time constraints.
- SAC is a very reliable and sample-efficient algorithm for continuous control tasks, which makes it a good fit for our problem. It also has built-in mechanisms for handling exploration and stability, which are important given the complexity of the environment and the need to balance safety and performance.
- A safety shield acts as a last line of defense between the drone and the various trees. Later versions of the project will include active dodging instead of breaking.

---

3. Environment and implementation

3.1 Simulation setup

- We use a custom fast in-memory simulator (`fastsim`) for training as mentioned above, making training orders of magnitude faster than using Gazebo directly. This allows us to iterate quickly on our policy and reward design, and to run many parallel environments for efficient training. For evaluation and demo purposes, we use Gazebo+ROS2 to run rollouts in more realistic environments that closely mimic the conditions the UAV will face in the real world.
- Each episode lasts until either the UAV reaches the goal, collides with an obstacle, or exceeds a maximum time limit (e.g., 90 seconds, defined in the .yaml configs).
- We use a configurable stochastic world generator to create diverse, realistic forest environments for training and evaluation (`worldgen`). The generator can create worlds with varying tree densities, placements, and types, allowing us to train a policy that can generalize across different forest conditions.

3.2 Observations

- The observation space is a 1D vector of size `N + 6`, where `N = lidar_num_beams`.
- In training/Gazebo, `N = 90`, so the observation dimension is 96.
- The first `N` components are LiDAR ranges normalized as `clip(range / lidar_range_max, 0, 1)`.
- The final 6 components are:
  - `cos(theta_goal)`, where `theta_goal` is the goal direction in the UAV body frame,
  - `sin(theta_goal)`,
  - normalized goal distance: `clip(dist_to_goal / (2 * world_half_extent), 0, 1)`,
  - normalized forward speed: `clip(v / v_max, -1, 1)`,

    - normalized yaw rate: `clip(wz / wz_max, -1, 1)`,
    - normalized height error: `clip((z_target - z) / z_error_scale, -1, 1)`.
- The environment declares `observation_space = Box(low=-1, high=1, shape=(N+6,), dtype=float32)`.

3.3 Action space and control interface

Describe outputs and how they are converted to commands: - The action space is a 3D continuous vector representing the desired forward acceleration, yaw angular aggeneration, and vertical acceleration: `action = [ax, ayaw, az]`. - The policy outputs these values in a normalized range (e.g., `[-1, 1]`), which are then scaled to the actual control limits of the UAV (e.g., `ax_scaled = ax * ax_max`, `ayaw_scaled = ayaw * ayaw_max`, `az_scaled = az * az_max`). - These scaled commands are then sent to the simulator or the real UAV's flight controller, which after passing them through the safety shielf filter, executes them at a fixed control frequency (e.g., 20 Hz).

3.4 Safety shield

- **What the shield checks:** minimum clearance to obstacles (LiDAR ranges and nearest-point distance), estimated time-to-collision / imminent-collision prediction using range rate and current velocity, kinematic and dynamic constraints (max deceleration, max yaw rate), and mission constraints (height and speed limits).
- **What it outputs:** modified or overridden commands such as clamping forward and vertical accelerations, scaling down or zeroing forward thrust to command a safe stop/hover, commanding a yaw-away maneuver to steer clear of imminent collisions, and a binary intervention flag for logging and evaluation. Outputs are smoothed to avoid abrupt discontinuities.
- **Where it runs in the loop:** executed immediately after the policy produces actions and before sending commands to the simulator or flight controller (Sensors -> Observation -> Policy -> Safety shield -> Command -> Simulator/Robot). It operates at the control loop rate (e.g., 20 Hz) to provide low-latency, deterministic filtering of actions.

3.5 Reward design

High-level description (omitting quite a few details). - **Progress reward:** positive reward proportional to the reduction in distance-to-goal between consecutive steps (or the projection of the movement onto the goal direction). Encourages goal-directed motion and provides dense learning signal. - **Speed incentive:** a modest positive term for forward speed when the robot is oriented toward the goal (e.g., positive only when velocity projects positively on goal direction). Encourages efficient traversal and reduces time-to-goal while avoiding incentives to speed blindly into obstacles. Maximized when speed = max speed. - **Proximity penalty:** a negative term that increases as the closest LiDAR range decreases (for example, inverse-distance or clipped exponential). Encourages maintaining clearance from obstacles and produces graded avoidance behavior rather than only relying on collision signals. - **Step penalty:** a small constant negative reward per step to discourage dithering, circling, or overly long trajectories and to bias the agent toward quicker solutions. - **Collision penalty:** a large negative terminal reward given on collision to strongly disincentivize contact and make collisions clearly distinguishable from other failures.

---

4. Training methodology

4.1 Algorithm

- **Algorithm:** Soft Actor-Critic (SAC) for continuous control (Stable-Baselines3-style configuration).
- **Policy:** `MlpPolicy` with MLP architecture `[512, 512]` for actor and critics (ReLU activations).
- **Learning rates:** global learning rate = 0.0002; entropy coefficient (`ent_coef`) = `auto` (learned).
- **Replay buffer:** size = 1_000_000 transitions; `learning_starts` = 10_000 steps.
- **Batch size:** 1024.
- **Optimization schedule:** `train_freq` = 4 (env steps), `gradient_steps` = 8, `target_update_interval` = 1.
- **Discount & targets:** gamma = 0.99, target smoothing `tau` = 0.005.
- **Other:** `policy_kwargs.net_arch = [512, 512]`, `verbose` enabled for training runs.

4.2 Training setup

- **Vectorized environments:** `n_envs` = 16 parallel `fastsim` instances for data collection.
- **Total timesteps:** `total_timesteps` = 10_000_000 for production runs; shorter runs used for development.
- **Random seeds:** canonical runs use `seed` = 42; report variance by repeating runs across seeds when required.
- **Domain randomization:** randomized tree radii/density, spawn jitter, sensor noise and small actuation noise (see `env` params) to improve sim-to-real robustness.
- **Normalization & clipping:** observation and reward normalization enabled (VecNormalize) with `clip_obs` = 10.0 and `clip_reward` = 10.0.
- **Action scaling & safety:** actions are learned in normalized space and scaled to actuators; the external safety shield filters actions at control rate and is logged separately.

4.3 Logging and checkpoints

- **Logging:** TensorBoard summaries and CSV episode logs written to `code/forest_navigating_uav/outputs/runs/<run_id>`; logged metrics include reward, success, collisions, shield interventions, and losses.
- **Evaluation frequency:** deterministic evaluation runs every `eval_freq_step` = 50_000 steps with `n_eval_episodes` = 10.
- **Checkpointing:** periodic checkpoints every `checkpoint_freq_step` = 100_000 steps; a "best" checkpoint (by eval success rate) is kept. Replay buffer saving is configurable.
- **Saved artifacts:** model weights, optimizer states, VecNormalize statistics, training YAML, and evaluation rollouts/videos for reproducibility.

---

5. Evaluation protocol

5.1 Evaluation scenarios

We evaluate policy in 3 ways - Periodic evaluation during training in `fastsim`. - `make rl-trajectories` after training to visualize fastsim performance. - Gazebo+ROS2 rollouts in custom forest worlds for sim-to-real validation and demo purposes.

5.2 Metrics (definitions)

- Success = percentage of episodes where the UAV reaches the goal without collisions.
- Collision = percentage of episodes where the UAV collides with a tree.
- Time to goal = time (in seconds) taken to reach the goal in each episode.
- Minimum clearance = minimum distance (in meters) between UAV and any tree during an episode.
- Shield intervention rate = percentage of steps where the external safety shield intervened.

---

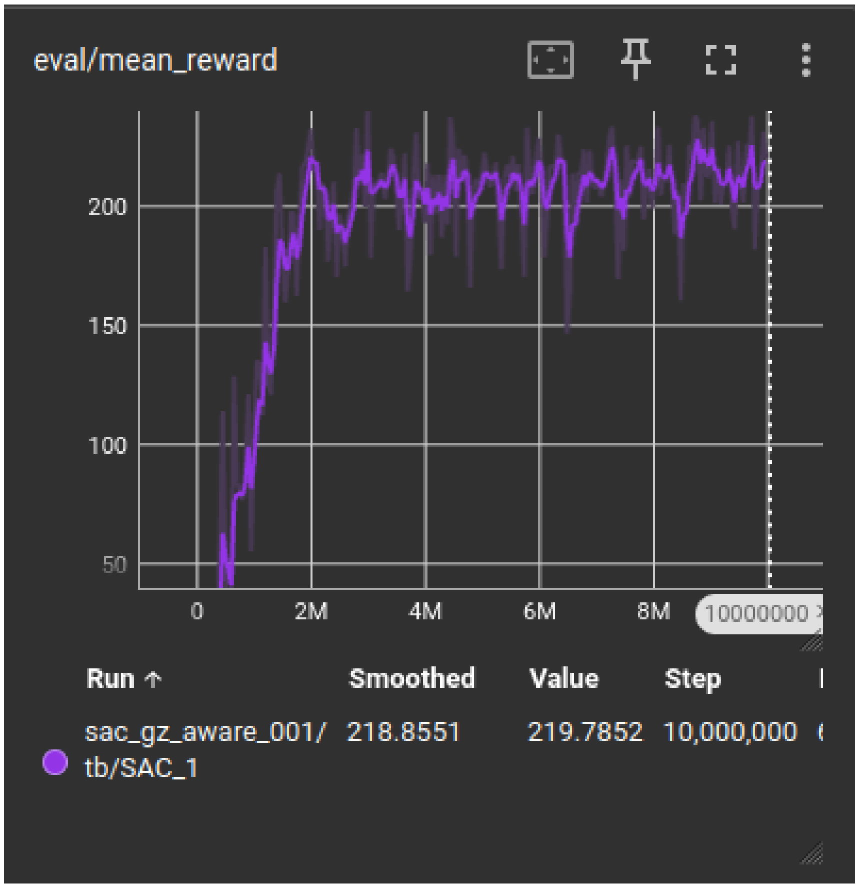
6. Results

6.1 Quantitative results (main table)

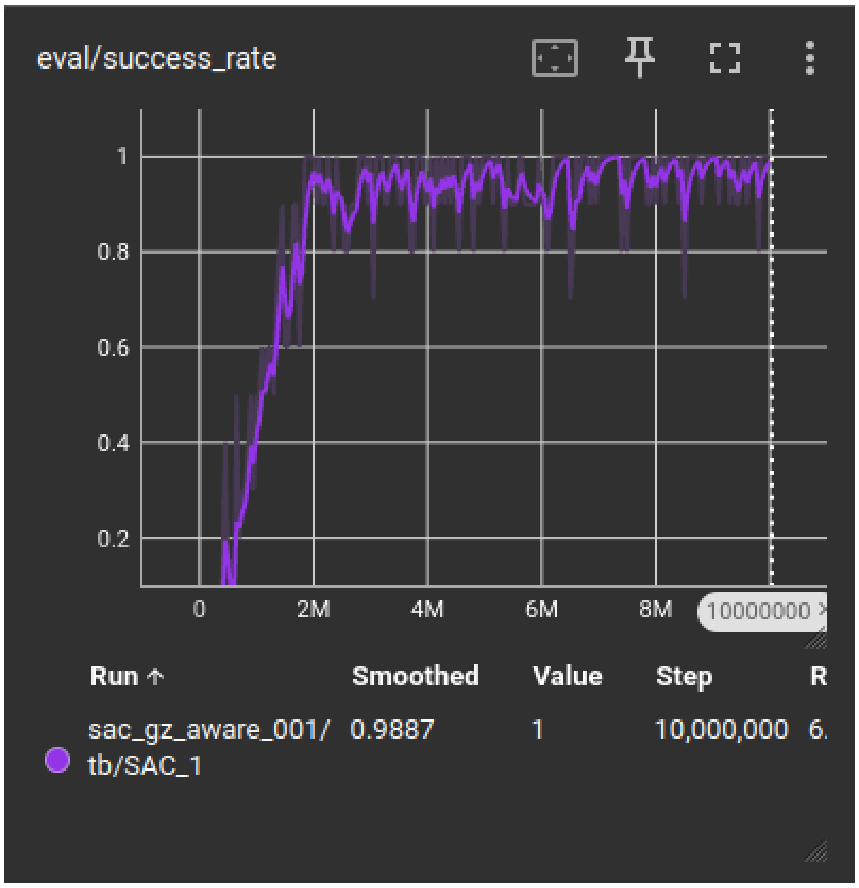
Metric	Baseline	Final	Delta
Success rate (%)		85.0	
Collision rate (%)		11.95	
Median time to goal (s)		12.35	
Min clearance (m)		2.806	
Shield interventions (% steps)		8.429	

6.2 Learning curves and plots

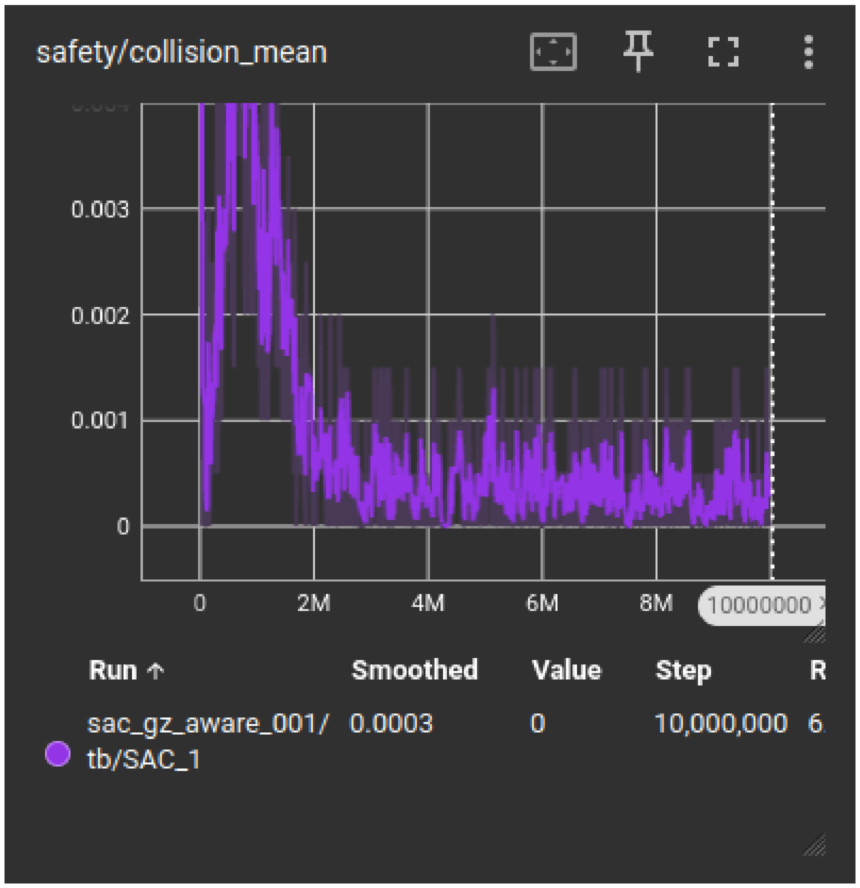
- Reward curve:



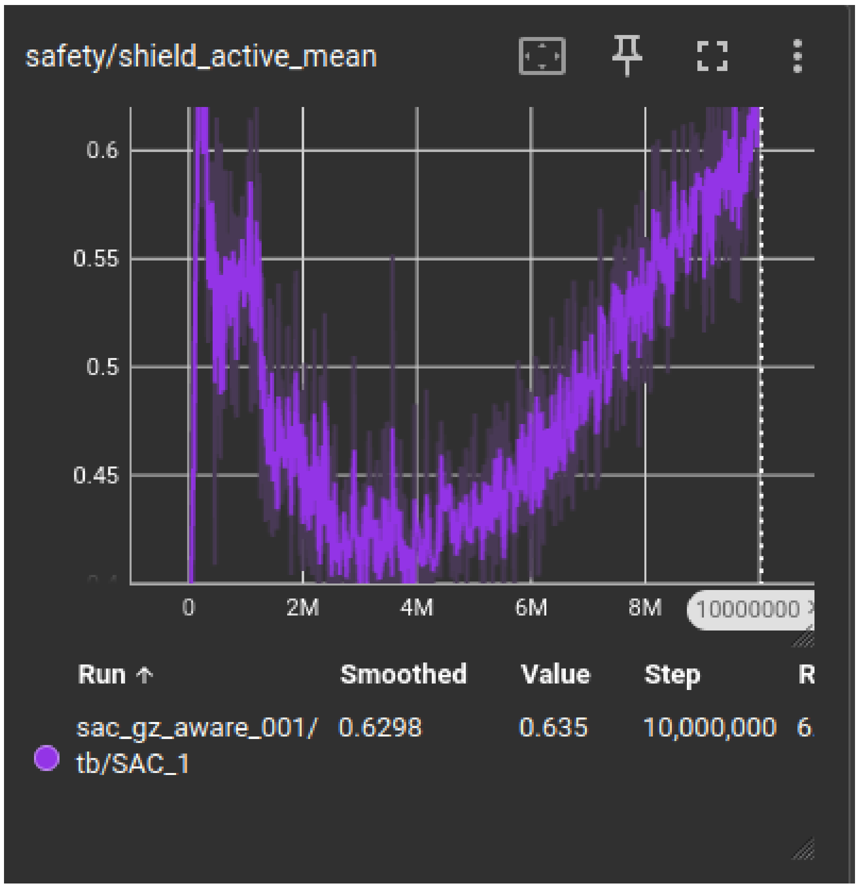
	Run ↑	Smoothed	Value	Step
●	sac_gz_aware_001/ tb/SAC_1	218.8551	219.7852	10,000,000

- Success rate: 

Run ↑	Smoothed	Value	Step	R
● sac_gz_aware_001/ tb/SAC_1	0.9887	1	10,000,000	6.

- Collisions: 

	Run ↑	Smoothed	Value	Step	R
●	sac_gz_aware_001/ tb/SAC_1	0.0003	0	10,000,000	6.

- Shield interventions: 

Run ↑	Smoothed	Value	Step	R
● sac_gz_aware_001/tb/SAC_1	0.6298	0.635	10,000,000	6

---

7. Demo and deliverable media

7.1 Videos

Your browser does not support the video tag. Download the video from media/videos/autonomous_agents_long_rendered_compressed.mp4.

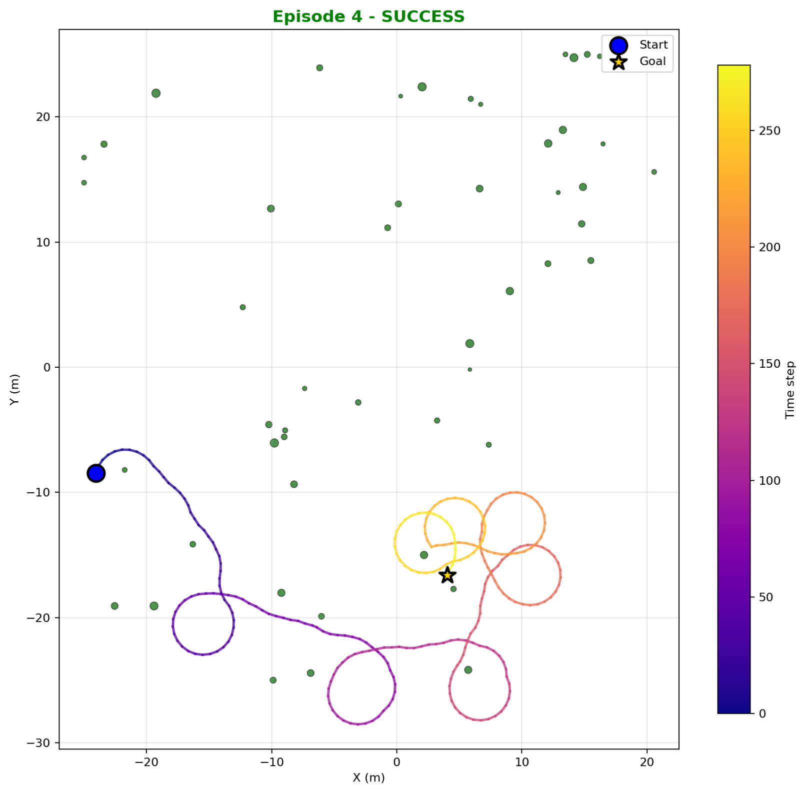
---

8. Technical difficulties and fixes

Issue 1:

- Symptom: Technically correct but undesirable behavior (e.g. agent doing circles, or in another case, travelling outside the forest).
- Root cause: Reward design that incentivizes forward speed without sufficient penalty for proximity to obstacles, leading to reckless behavior.

- Fix: Fine tuning the reward function, adding a virtual barrier that would count as a collision around the trees
- Why it worked: doing doughnuts around trees to “farm” speed rewards is not worth it anymore



- Behavior before finetuning, I was happy at the time because it was the first episode success I had ever gotten.

Issue 2:

- Symptom: Policy trained in `fastsim` performed poorly in Gazebo, with a high collision rate and erratic behavior.
- Root cause: The `fastsim` environment had simplified dynamics and sensor models that did not capture the full complexity of Gazebo’s physics and noise characteristics, leading to a sim-to-sim transfer gap. Also, gazebo is a lot more junky in general, leading to very weird behavior
- Fix: Changed the Gazebo config to match the `fastsim` setup more closely, froze height, roll and pitch (ignoring policy z acceleration action output)

Issue 3:

- Symptom: Training was very slow and required a lot of compute resources, making it difficult to iterate on the design and run multiple experiments.
- Root cause: The combination of a complex environment, a large replay buffer, and a constant need for various checks, all of which are CPU-bound
- Fix: `forest_nav_env.py` implements a hash grid of tree indices for more efficient raycasting, lazy vectorized collision checking, parallelized environment stepping and reward computation, and other optimizations to reduce the computational load and speed up training.

Issue 4:

- Symptom: The `fastsim` environment's simplified dynamics led to unrealistic behavior and poor generalization to Gazebo and real-world conditions.
- Root cause: Initial fastsim/policy designs dealt in `velocity` commands, resulting in "UFO" performance with no inertia or momentum, which is unrealistic for a UAV and led to poor transfer.
- Fix: Redesigned the action space to output accelerations instead of velocities. This is still suboptimal, and eventually the simulaotr will be refined to use thrust commands and a wider range of movement axes.

---

9. Discussion

9.1 What worked and why

- The learned policy was able to effectively navigate the complex forest environment and reach the goal with a high success rate, demonstrating that a data-driven approach can capture the necessary patterns and behaviors for this task.
- The safety shield successfully prevented collisions in many cases, allowing the policy to take more aggressive actions without risking catastrophic failure, which contributed to the improved performance and safety of the system.
- The use of a custom fast in-memory simulator for training allowed for rapid iteration and efficient data collection, which was crucial for developing a robust policy within a reasonable timeframe. The domain randomization techniques also helped improve the policy's generalization capabilities, as evidenced by its performance in the more realistic Gazebo environment.

9.2 Limitations

Be honest, but controlled. - The policy's performance in Gazebo, while promising, still showed a significant gap compared to `fastsim`, indicating that there are still sim-to-sim transfer issues that need to be addressed before real-world deployment. - The safety shield, while effective at preventing collisions, can be overly conservative in some scenarios, leading to unnecessary interventions that reduce the overall efficiency of the system. This is particularly evident in scenarios with narrow passages where the shield may prevent the UAV from taking necessary actions to navigate through tight spaces. - The current action space and dynamics model are still quite simplified, and do not capture the full range of behaviors and constraints that a real UAV would face, such as wind disturbances, more complex aerodynamics, and the need for more precise control over roll and pitch angles. This limits the realism of the training environment and the potential for successful sim-to-real transfer.

---

10. Conclusion and future work

- We developed a learning-based navigation system for UAVs in cluttered forest environments, achieving an 85% success rate in reaching goals without collisions in a custom simulator.
- We designed and implemented a safety shield that effectively prevented collisions, contributing to the improved safety and performance of the system.
- We demonstrated the potential for sim-to-sim transfer by evaluating the trained policy in Gazebo, although there is still work to be done to close the gap between the two environments.
- We identified several technical challenges and limitations in our approach, including reward design issues, sim-to-sim transfer difficulties, computational intensity, and the need for more realistic dynamics modeling.
- 3 to 6 bullets for next steps (Gazebo transfer, better dynamics, better perception, better safety proofs)
- Future work will focus on improving the realism of the training environment, refining the safety shield to be less conservative while still ensuring safety, and ultimately testing the system on real UAV hardware in controlled forest environments to validate its performance and robustness in real-world conditions.
- This A -> B movement is only a movement primitive, and future work will also explore more complex tasks such as multi-goal navigation, dynamic obstacle avoidance, and coordination in multi-agent scenarios.
- Additionally, I'll try to train a model in an exponentially more complex environment (with bushes, branches, etc)
- Implementing Computer Vision-based perception to complement or replace LiDAR sensing, enabling the UAV to better understand and navigate complex forest environments with occlusions and varying lighting conditions will be interesting to explore as well.

- Lastly, I will try my hand at multi-agent systems, utilizing SLAM and communication between agents to enable cooperative navigation and task completion in forest environments.