Test Case Generation for Autonomous UAV Navigation using Q-learning and UCB

Ali Javadi*

Abstract

The testing of autonomous UAV systems poses significant challenges due to the complexity of setting up realistic, diverse test scenarios. This report documents a method that applies Q-learning combined with Upper Confidence Bound (UCB) to generate obstacle configurations in a simulated environment. The goal is to evaluate the PX4-Avoidance system by inducing unsafe UAV behaviors through generated obstacle placements. This approach enhances fault detection by balancing exploration and exploitation in the test case generation, ultimately increasing scenario diversity and system robustness.

1 Introduction

Simulation-based testing has emerged as an effective means of uncovering UAV vulnerabilities prior to field testing [1, 2]. Complex UAV systems require a robust testing approach to ensure the software handles diverse scenarios, especially for vision-based autonomous obstacle avoidance systems. This study implements an automated test generation tool utilizing Q-learning [3] with an Upper Confidence Bound (UCB) [4] for balancing exploration-exploitation. The approach generates challenging environments with obstacles, aiming to increase test diversity and failure detection efficiency.

*This report was initially drafted independently and subsequently revised using AI language tools (Grammarly, Google Gemini, and ChatGPT) to enhance grammatical accuracy and overall readability.

2 Methodology

The algorithm is based on reinforcement learning, specifically Q-learning, to select actions for modifying obstacles in a simulated environment. Using UCB, we control the exploration of less-tested actions, ensuring effective coverage of potential failure scenarios.

2.1 Problem Formulation

The test environment represents a predefined area containing up to three obstacles of varying sizes and orientations, aimed at disturbing the PX4-Avoidance system's navigation. Each test case is evaluated on fault detection and diversity, where failures are categorized as hard (collision) or soft (unsafe proximity) fails. The algorithm aims to maximize these failure detections through a constrained budget of test attempts.

2.2 Q-learning

Q-learning, a model-free reinforcement learning algorithm, is employed to maximize the rewards associated with generated test scenarios. The Q-value update equation, as presented in [3], is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1}$$

where s represents the state, a the action, α is the learning rate, γ the discount factor, r the reward, and $Q(s,a)$ is the Q-value for a state-action pair.

2.3 Upper Confidence Bound (UCB) Strategy

To balance exploration and exploitation, UCB is applied to determine the best action in each state, as defined in [4]:

$$UCB = Q(s,a) + c\sqrt{\frac{\ln(N)}{N(a)}} \tag{2}$$

where c is an exploration parameter, N the total number of attempts, and $N(a)$ the count of actions taken in that state. This incentivizes less-tried actions, encouraging broader exploration.

2.4 Reward Calculation

The reward function is defined as:

$$r = \begin{cases} 5 + (3 - n) & \text{if } d_{\min} < 0.25 \\ 2 + (3 - n) & \text{if } 0.25 \leq d_{\min} < 1 \\ 1 + (3 - n) & \text{if } 1 \leq d_{\min} < 1.5 \\ 0 & \text{otherwise} \end{cases}$$

where n is the number of obstacles, and $d_{\min}$ (in meters) is the minimum distance of the trajectory to any obstacle. Fewer obstacles yield higher rewards to encourage simpler, more effective failure cases.

2.5 Pseudocode for Test Generation

The pseudocode for the generator is shown below:

Algorithm 1 UCB-based Test Case Generation

```
Initialize Q(s,a) table, action counts
for each iteration i = 1 to budget do
    Sample initial obstacle configuration o
    s ← get_state(o)
    a ← choose_action(s, actions, total_attempts)
    Apply action a to obstacles, ensuring no overlap
    Execute test case and record minimum distance d_min
    Calculate reward based on d_min
    Update Q-table: Q(s,a) ← Q(s,a) + α(r + γ max_a Q(s',a) − Q(s,a))
end for
```

3 Implementation Details

The generator creates a random obstacle within the size and position ranges, ensuring no overlap among obstacles. If overlapping occurs, adjustments are made based on chosen actions. The ‘calculate_reward‘ function assigns a higher reward for scenarios with fewer obstacles that result in failures. This reward system aligns with the competition's goal of prioritizing simpler, effective failures. The competition emphasizes that faults found in less complicated environments (with fewer obstacles) will be valued more highly.

3.1 State Representation

The test generator represents each obstacle configuration as a state, encoding information about the position, size, and orientation of each obstacle. The 'get_state' function converts these obstacle attributes into a state tuple used for decision-making.

3.2 Actions and Rewards

Four primary actions adjust the obstacle configuration: 'adjust_x', 'adjust_y', 'resize', and 'rotate'. The ‘choose_action‘ method evaluates each action's UCB value to maintain exploration, whereas the ‘calculate_reward‘ function bases rewards on UAV proximity to obstacles.

4 Experimental Results

The generated test cases are evaluated based on the reward function defined in Section 2.4 and the fitness function given in [1]. The fitness function is designed to guide the algorithm to bring the drone close to all obstacles in general, and particularly close to the edge of one specific obstacle. This guidance encourages scenarios that test the UAV's ability to navigate around obstacles while inducing risky behavior.

The fitness function is defined as follows:

$$sum\_dist = \min_{p \in trj.points} \sum_{o \in obs} d(p,o)$$

$$min\_dist = \min_{o \in obs, p \in trj.points} d(p,o)$$

$$fitness = sum\_dist + 2 \times min\_dist$$

In this formulation, $d(p, o)$ represents the distance between a trajectory point p and an obstacle o, $trj.points$ refers to the set of points in the trajectory, and obs denotes the set of obstacles.

4.1 Experiment Setup

The experiments were conducted within Docker containers on a system equipped with 16 GB of memory and a "13th Gen Intel® Core™ i5-13420H" processor with 12 cores. The test parameters were configured as follows: the total test attempt budget, α, γ, and c were set to 100, 0.5, 0.99, and $1/\sqrt{2}$, respectively. These values were chosen to optimize the balance between exploration and exploitation during the test case generation process.

The test generator was evaluated across three distinct missions from the UAV Testing Competition. Each mission aimed to challenge the quadcopter's navigation capabilities through varying obstacle configurations. The unfinished and unsafe rate were measured to assess the effectiveness of the generated test scenarios.

4.2 Evaluation

I tested the test generator on three different missions provided in the UAV Testing Competition, which can be found at 'https://github.com/skhatiri/UAV-Testing-Competition'.

The results of the test generator across three distinct missions are shown in Table 1. This table summarizes the rate of unfinished missions and instances of unsafe proximity (minimum distance to obstacles less than 1.5 meters) for each mission.

Table 1: Mission Results: Unfinished and Unsafe Missions (out of a hundred)

Mission	Unfinished Rate	Unsafe Rate
Mission 1	0.09	0.02
Mission 2	0.09	0.04
Mission 3	0.08	0.04

Figure 1 illustrates instances of the diverse obstacle configurations generated by the test generator, and Figure 2 demonstrates some scenarios in which the drone failed to reach the goal. The analysis of these results demonstrates the generator's capability to produce challenging scenarios that effectively test the quadcopter's navigation abilities.

4.3 Discussion

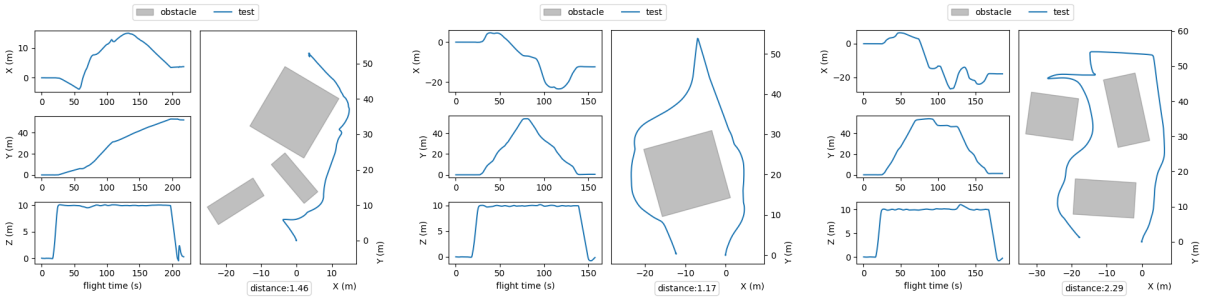
After experimenting with different operators in the reward function, ranging from low steepness (addition) to high steepness (exponentiation), we observed a potential correlation between the steepness of the reward function and the rates of failure and unsafe outcomes in test generation, as shown in Table 2. Additionally, Figure 3 illustrates that, compared to the addition-based reward function (Figure 3.a), the multiplication-based (Figure 3.b) reward function exhibits larger ripples, while the exponentiation-based (Figure 3.c) reward function shows more frequent ripples with similar magnitude. I believe this presents an intriguing avenue for further investigation.

Table 2: Impact of Reward Function Operators on Mission 3 Outcomes (out of a hundred)

Operator	Unfinished Rate	Unsafe Rate
Addition (+)	0.08	0.03
Multiplication (×)	0.15	0.04
Exponentiation (^)	0.12	0.08

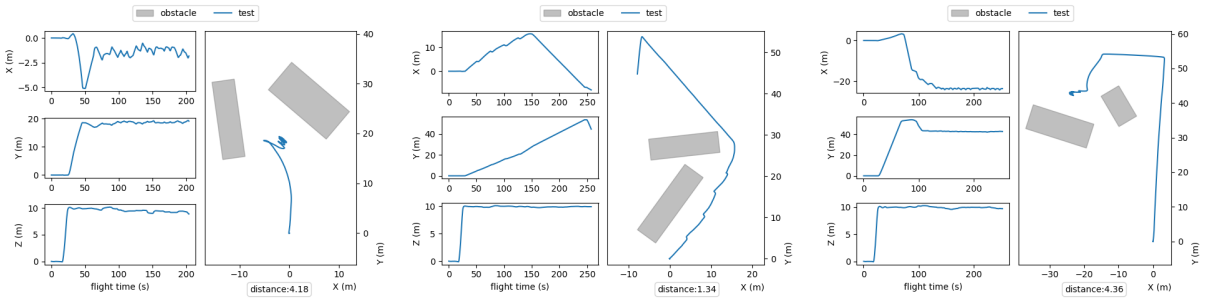
5 Conclusion

The Q-learning and UCB-based test generator effectively maximizes UAV failure detection and scenario diversity by balancing exploration and exploitation. This method demonstrates potential for scalable, simulation-based UAV testing with robust adaptability to complex environments. Future work may involve further optimization of the reward structure and adaptation to different UAV models and obstacle settings.



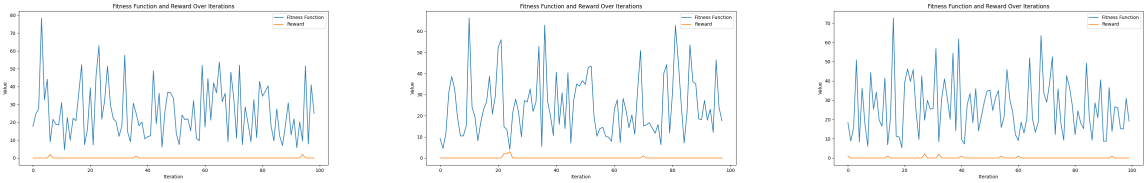
(a) Mission 1 (b) Mission 2 (c) Mission 3

Figure 1: Obstacle configurations generated for different missions in the UAV Testing Competition.



(a) Mission 1 (b) Mission 2 (c) Mission 2

Figure 2: Failure cases generated by test generator across different missions.



(a) Addition (b) Multiplication (c) Exponentiation

Figure 3: Impact of different operators on reward function outcomes in Mission 3.

References

[1] Khatiri, Sajad, Sebastiano Panichella, and Paolo Tonella. "Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights." 2023 IEEE Conference on Software Testing, Verification and Validation (ICST). IEEE, 2023.

[2] Khatiri, Sajad, Sebastiano Panichella, and Paolo Tonella. "Simulation-based testing of unmanned aerial vehicles with aerialist." Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings. 2024.

[3] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8 (1992): 279-292.

[4] Auer, Peter. "Using confidence bounds for exploitation-exploration trade-offs." Journal of Machine Learning Research 3.Nov (2002): 397-422.