

# RLaGA: A Reinforcement Learning Augmented Genetic Algorithm For Searching Real and Diverse Marker-Based Landing Violations

LINFENG LIANG, Macquarie University, Australia

YAO DENG, Macquarie University, Australia

KYE MORTON, Skyy Network, Australia

VALTTERI KALLINEN, Skyy Network, Australia

ALICE JAMES, Macquarie University, Australia

AVISHKAR SETH, Macquarie University, Australia

ENDROWEDNES KUANTAMA, Macquarie University, Australia

SUBHAS MUKHOPADHYAY, Macquarie University, Australia

RICHARD HAN, Macquarie University, Australia

XI ZHENG\*, Macquarie University, Australia

Automated landing for Unmanned Aerial Vehicles (UAVs), like multirotor drones, requires intricate software encompassing control algorithms, obstacle avoidance, and machine vision, especially when landing markers assist. Failed landings can lead to significant costs from damaged drones or payloads and the time spent seeking alternative landing solutions. Therefore, it's important to fully test auto-landing systems through simulations before deploying them in the real-world to ensure safety. This paper proposes RLAGA, a reinforcement learning (RL) augmented search-based testing framework, which constructs diverse and real marker-based landing cases that involve safety violations. Specifically, RLAGA introduces a genetic algorithm (GA) to conservatively search for diverse static environment configurations offline and RL to aggressively manipulate dynamic objects' trajectories online to find potential vulnerabilities in the target deployment environment. Quantitative results reveal that our method generates up to 22.19% more violation cases and nearly doubles the diversity of generated violation cases compared to baseline methods. Qualitatively, our method can discover those corner cases which would be missed by state-of-the-art algorithms. We demonstrate that select types of these corner cases can be confirmed via real-world testing with drones in the field.

**CCS Concepts:** • Software and its engineering → Search-based software engineering

**Additional Key Words and Phrases:** UAV auto-landing system, Genetic Algorithm, Reinforcement Learning, Search-based testing

---

\*Corresponding authors: Xi Zheng.

Authors' addresses: Linfeng Liang, Linfeng.liang@hdr.mq.edu.au, Macquarie University, Sydney, NSW, Australia, 2113; Yao Deng, yao.deng@hdr.mq.edu.au, Macquarie University, Sydney, NSW, Australia, 2113; Kye Morton, kye@skyy.network, Skyy Network, Sydney, NSW, Australia, 2113; Valtteri Kallinen, valtteri@skyy.network, Skyy Network, Sydney, NSW, Australia, 2113; Alice James, alice.james@hdr.mq.edu.au, Macquarie University, Sydney, NSW, Australia, 2113; Avishkar Seth, avishkar.seth@mq.edu.au, Macquarie University, Sydney, NSW, Australia, 2113; Endrowednes Kuantama, endrowednes.kuantama@mq.edu.au, Macquarie University, Sydney, NSW, Australia, 2113; Subhas Mukhopadhyay, subhas.mukhopadhyay@mq.edu.au, Macquarie University, Sydney, NSW, Australia, 2113; Richard Han, richard.han@mq.edu.au, Macquarie University, Sydney, NSW, Australia, 2113; Xi Zheng, xi.zheng@mq.edu.au, Macquarie University, Sydney, NSW, Australia, 2113.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Association for Computing Machinery.

XXXX-XXXX/2023/10-ART \$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

### ACM Reference Format:

Linfeng Liang, Yao Deng, Kye Morton, Valtteri Kallinen, Alice James, Avishkar Seth, Endrowednes Kuantama, Subhas Mukhopadhyay, Richard Han, and Xi Zheng. 2023. RLaGA: A Reinforcement Learning Augmented Genetic Algorithm For Searching Real and Diverse Marker-Based Landing Violations. 1, 1 (October 2023), 19 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

Automated landing of UAVs, namely multirotor drones, is already a standard software mode used in today's drone flight controllers [1], often invoked during the last stages of flight when a drone has spotted a marker on the ground for landing. The usage of this capability is expected to increase rapidly as an expanding number of autonomous UAV services emerge, such as automatic drone monitoring, surveying, and package delivery [31, 37, 45], in which many drones are not manually controlled. The landing procedure for drones is critical during flight operations as there is a high risk of colliding with mobile ground units such as pedestrians and animals. The addition of ground-truth markers can significantly reduce the risk of accidents [5, 21, 30]. However, marker-based landing can be disturbed by many factors such as weather, on-board sensor limitations, the potential for false identifications causing landing in incorrect locations or missed landing in correct locations, and/or moving objects on the ground which may cause more serious accidents during landing [34]. Airspace regulators, such as CASA [4] and the FAA [3], require strict safety and separation requirements to perform commercial activities in public spaces. The difficulty of ensuring autonomous UAV meet these requirements, along with recent incident reports, highlight the importance of reducing risk throughout all steps of UAV landing procedures [8, 46]. This would prevent accidents, such as those caused by dynamic objects moving around the marker, and would significantly increase the robustness of these systems in the most dangerous stages of flight. Therefore, comprehensive tests must be conducted on the automated drone landing software to support safe UAV deployment, especially in anticipation of widespread autonomous UAV operation.

Simulation testing is commonly used to assess Autonomous Driving systems (ADSs) [10, 15, 24, 26]. UAV operations, including marker-based UAV landings, can be tested in a similar manner using a comprehensive simulation environment such as AirSim [36]. Recent studies indicate that search-based methods such as Genetic Algorithms (GA) or Reinforcement Learning (RL) are promising in testing the AI-based cyber-physical systems [18, 26, 28]. However, GA relies on mutation and crossover and tends to maintain populations with high fitness values, which can lead to GA's inclination to converge towards local optima. As a consequence, this approach may result in similar found test cases [14]. Conversely, using RL for entirely online searches can potentially induce rapid scenario alterations, risking unrealistic generated test cases [15, 27]. An ideal search method should be able to cover all possible violations in the real-world. Therefore, the diversity and realism of generated test cases in the simulation are important.

A novel RL-augmented GA (RLaGA) method is proposed to address these challenges. Figure 1 indicates a high-level overview of our framework. Our method uses GA to conservatively search for environment configuration offline and RL to aggressively search for the movement of dynamic objects online to generate diverse and realistic corner cases in a marker-based landing system. Specifically, our GA generates diverse environment configurations, while RL controls the dynamic objects within the environment to interact with the auto-landing-enabled UAV. Through our experiments, we demonstrate how our integrated approach effectively addresses the aforementioned challenges. Our contributions can be summarised as follows:

- We propose a novel integrated RL-augmented GA (RLaGA) method to efficiently generate diverse and realistic corner test cases for marker-based UAV landing systems. We introduce a customized offline GA using our chromosome representation with two types of genes to

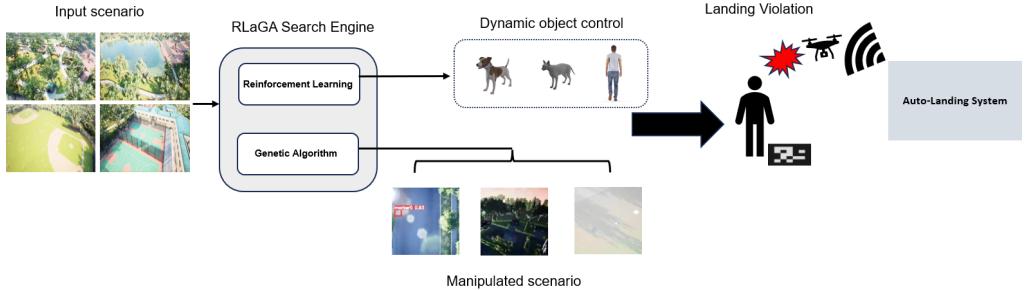


Fig. 1. A high-level overview of RLaGA framework, including the simulator and the search engine.

control conditions such as weather and RL-guided online manipulation of dynamic objects in the scene.

- We demonstrate through extensive experiments in simulation that RLAGA discovers up to 22.19% more violations and nearly doubles the diversity of generated violation cases in comparison to state-of-the-art baselines in different map settings.
- We confirm that we are able to reproduce in the real-world select types of the landing violations found in simulation by flying drones in practical environments while running automated landing software.

This paper is organized as follows: In section 2, we provide a review and discussion of the related work relevant to our study. Section 3 details how we created our search method. In section 4, we indicate the design of our experiments. In section 5, we present the results of our comprehensive experiments based on the methodology described in section 3. In section 6, we discuss potential threats against our proposed methods. Conclusions are summarized in section 7.

## 2 RELATED WORK

The efficacy of GA has been demonstrated in searching for corner test cases that cause violations and failures in AI-based cyber-physical systems [24, 32, 35]. GA has extensive applications in search-based software testing. GA is usually employed as an offline search technique [47]. AV-FUZZER [24] employs both a global fuzzer and a local fuzzer based on GA to search for corner cases in ADSs. StellaUAV [35] uses GA and different optimisation methods to search for corner cases in the dynamic obstacle avoidance system of UAVs. AutoFuzz [49] demonstrates that neural networks have the potential to augment GA and employs a gradient as an indicator to mutate seeds. Similar research has been conducted to demonstrate that GA can also search for multiple objectives in ADS testing [2, 16, 32, 42]. A common issue is that GA can converge on a local optimal solution during the search. We propose a novel approach that involves a GA with intentionally designed mutations and crossover operations to prevent this. The objective of this approach is to enhance the diversity among the concrete violation scenarios in marker-based landing tests.

The validity of RL has been demonstrated in generating violation cases in software tests [33, 38]. RL is a Markov Decision Process (MDP) [41] which is built based on an interactive environment that includes agents, actions, policies and rewards. In the MDP, the agent perceives the current state and performs actions in the environment based on the policy to receive the reward. Similar to GA, the RL agent learns to dynamically modify the logical scenario to generate corner cases through a predefined reward function. RL-based search methods are commonly employed as online search methods i.e. allowing for the variation of logical scenarios in time with each run. Incorporating RL allows for dynamic modification of an object's attributes, such as the speed and direction of

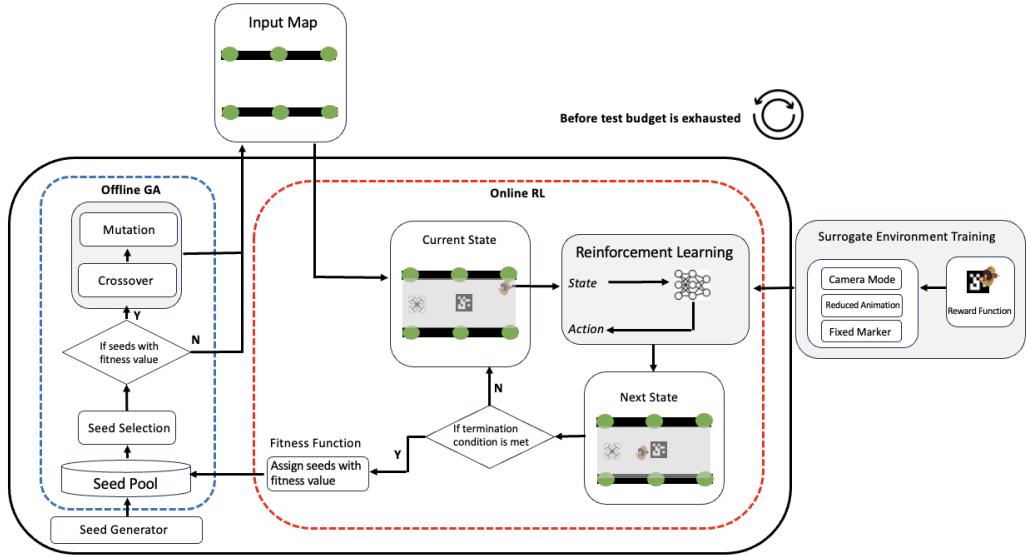


Fig. 2. Workflow of RLaGA

a person or animal. AST [9, 18] implements an RL that is able to update the environment of a vehicle to cause a collision. DEEPCOLLISION [28] demonstrates a DQN [43] approach that learns the configurations of the environment and can cause the crash of the ego vehicle. Research shows RL-based search methods can build reward functions from multiple objectives [15, 27]. This assists in refining the reward function and facilitates easier convergence of the RL agent.

The integration of RL with GA has been recently explored [12, 29]. Such methods test a cooperation method between RL and GA by generating corner cases for deep learning systems. Using RL to search for corner cases in simulation often leads to rapid changes in the simulated environment which can result in a lack of realism. In contrast, the method proposed in this paper utilises a novel GA-based RL to maintain the diversity and realism of generated corner cases.

### 3 METHOD

The search for violation cases starts with defining a logical scenario within the context of marker-based UAV auto-landing. Generally, the logical scenario is categorised into two main parts: environment configuration and dynamic objects. Environment configuration is searched offline by GA, which includes factors like weather parameters, the marker's position and the number and type of dynamic objects. These configurations remain constant throughout each run and are generated at the beginning of each run. Restrictions are imposed to ensure the generated parameters align with reality, where the specific restrictions vary depending on the type of parameter (Section 4.3). Dynamic objects are controlled online by the RL, and the RL agent will keep manipulating the dynamic objects' trajectories throughout each run.

Following this, we present an offline GA that is conservative in nature to explore diverse violation seeds. Subsequently, we propose an aggressive online RL algorithm that is designed to explore domain-expert-specified violation cases with high fidelity, while the RL can also benefit the GA. Figure 2 indicates the workflow of RLaGA. A seed generator initially initializes the seed pool and generates diverse seeds based on various deployment environments. It specifically considers

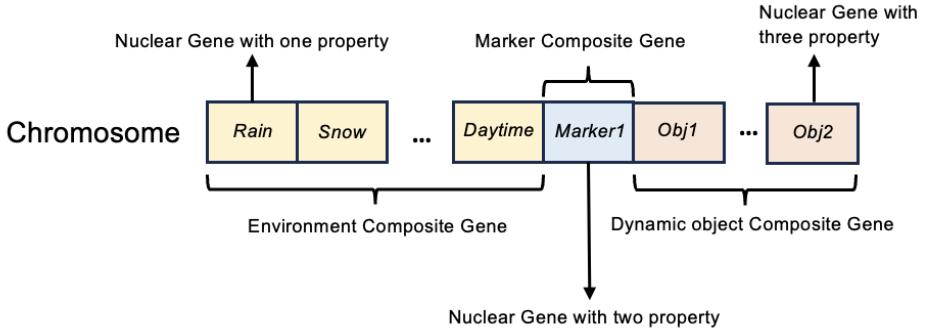


Fig. 3. The Chromosome Representation of Test Case.

the types of dynamic objects available that might affect UAV landing, as well as environmental conditions such as rain and weather. Each generation is then selected from the seed pool. If seeds have been assigned fitness values in prior generations, the GA is employed to form new representations offline for the subsequent generation; otherwise, the generation is executed in the deployment scenario. After initializing the scenario with the seed in the generation, the RL trained in a surrogate environment is engaged to online manipulate the trajectory of dynamic objects. Upon meeting the termination condition, the executed seed is assigned a fitness value and placed back into the seed pool to await selection for the next generation. This loop continues until the test budget is exhausted.

### 3.1 Genetic Algorithm - Offline

In this paper, we present a new genetic algorithm for identifying diverse violation seeds for marker-based UAV landings. We introduce a unique chromosome representation specifically for UAV landing scenarios, connecting offline GA and online RL. Our design features a tailored multi-objective fitness function for our UAV auto-landing scenario, and specialized crossover and mutation operations to enhance seed diversity. Further algorithm details are provided below.

**3.1.1 Chromosome Representation of Test Case.** Each test case in our method can be represented by a chromosome. Figure 3 indicates the chromosome representation of a test case.

We encode the environment, the marker position and available dynamic objects information in a given deployment environment in one chromosome, which consists of an *environment composite gene*, a *marker composite gene* and a *dynamic object composite gene*. The environment composite gene comprises single-scalar-property nuclear genes that denote the environment setting of specific environmental conditions in the test case, including *dust*, *fog*, *rain*, *snow*, *road wetness*, *falling maple leaf*, *leaf on the road*, *snow on the road*, and the position of the sun in the simulation environment. The marker composite gene uses a two-scalar-property nuclear gene to represent the marker's (x, y) position. The dynamic object composite gene has several three-scalar-property nuclear genes to specify each dynamic object's type, start point and velocity. A list of object types is specified by a given deployment scenario. For instance, for a deployment scenario, the available object types considered can be pedestrian, dog and bird. Moreover, dynamic object composite gene contains meta information to describe the count of each object type.

**3.1.2 Fitness Function.** Our multi-objective fitness function aims to identify varied scenarios that might compromise the safety of UAV landings. Given the perils associated with landing UAVs

in unmarked zones—areas that might hinder subsequent takeoffs or even lead to crashes—the primary objective addresses incorrect landing positions. Additionally, due to the significant power consumption of UAVs, extended landing durations can be risky. As such, our second objective focuses on the total time taken for landing. In line with these objectives, our fitness function, defined in Equation 1, incorporates two specific metrics: DistanceToLanding (DTL), which measures the deviation from the designated landing marker (objective 1), and TimeToLanding (TTL), which evaluates the required landing duration (objective 2).

$$FitnessValue = DTL + TTL \quad (1)$$

**3.1.3 Variation Operation.** The variation operators consist of crossover and mutation. We define a suite of variation operators to manipulate chromosomes after one generation. Algorithm 1 indicates our variation operators suite.

---

**Algorithm 1** The GA chromosome-based suite of variation operators

---

```

1: Input: Parents  $P$ , crossover threshold  $threshold_c$ , mutation threshold  $threshold_m$ , number of
   mutation candidate  $m$ 
2: Output: Population  $PN$ 
3:  $PN \leftarrow \emptyset$ 
4: for  $i$  in range( $0, |P|$ ) do
5:   select parent chromosome  $x_i \in P$ 
6:    $PN \leftarrow PN \cup x_i$ 
7: end for
8: generate  $r \sim U(0, 1)$ 
9: if  $r > threshold_c$  then
10:   for each two chromosomes  $x_i, x_j \in PN$  do
11:     generate separate point  $s \sim U(0, Len(x_i))$ 
12:      $x'_i, x'_j \leftarrow NuclearGeneCrossover(x_i, x_j, s)$ 
13:   end for
14: end if
15: for each chromosome  $x_i \in PN$  do
16:   for each nuclear gene  $y_i \in x_i$  do
17:     for each property  $y_{ij} \in y_i$  do
18:       generate  $r \sim U(0, 1)$ 
19:       if  $r > threshold_m$  then
20:          $M \leftarrow \emptyset$ 
21:         for  $i$  in range( $0, |m|$ ) do
22:           generate  $c \sim$  Property Range
23:            $M \leftarrow M \cup c$ 
24:         end for
25:          $y'_{ij} \leftarrow PropertyMutation(y_{ij}, M)$ 
26:       end if
27:     end for
28:   end for
29: end for
30: return  $PN$ 

```

---

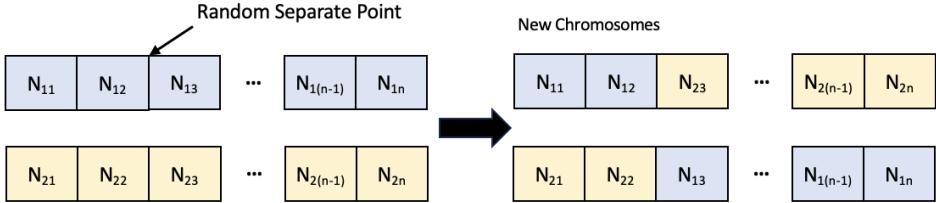


Fig. 4. Process of Nuclear Gene Crossover

**Nuclear Gene Crossover:** The crossover is a swap operation between two consecutive chromosomes that aims to swap the nuclear genes between two high-fitness value chromosomes to form new chromosomes that potentially have high fitness values. Figure 4 indicates the process of our crossover. If the crossover rate is met, our process begins by pairing two adjacent selected chromosomes. A random separation point within the chromosome length is chosen. The two paired chromosomes then disconnect from this point and crossover, forming two new chromosomes (line 8 to line 14 in Algorithm 1).

**Property-Level Mutation:** The primary issue is that conventional genetic algorithms tend to converge on local minima and find similar solutions [20, 24, 35]. Therefore, we design a mutation operation that would increase the diversity among chromosomes. Each property in the nuclear gene undergoes mutation according to the specified mutation rate (line 19 in Algorithm 1) which enhances exploration within the logical scenario. Specifically, if the mutation rate is met,  $m$  valid candidates for each property are randomly sampled based on available range (line 21 to 24 in Algorithm 1). Then, the candidate with the maximum distance to that property of all chromosomes in the seed pool is set as the value of the property in the current mutation chromosome (line 25 in Algorithm 1).

$$y'_{ij} = M \left[ \arg \max_{k \in K} |M[k] - Y_{ij}| \right] \quad (2)$$

Equation 2 indicates our core mutation strategy, where  $y'_{ij}$  is a  $i$ -th nuclear gene's  $j$ -th property in the chromosome after mutation,  $M$  is the mutation candidates set,  $K$  is the total number of mutation candidates,  $Y_{ij}$  is a set containing all chromosomes'  $i$ -th nuclear gene's  $j$ -th property in the seed pool.

### 3.2 Reinforcement Learning - Online

In the context of marker-based UAV landing, nearby objects can unintentionally disrupt the UAV's landing, leading to violations. While GA-driven objects lack precise online control capabilities, introducing RL can amplify the chances of detecting such violations. Using the chromosomes derived from the GA as a foundation, the RL dynamically controls these objects defined in the dynamic composite gene within our simulation environment. Rather than sampling pre-defined trajectory for dynamic objects in the integrated GA-RL method [42], RL guides the dynamic objects in a direction that heightens the probability of violations using a reward function that reflects the percentage of the UAV camera's field of view that has been obscured. However, online testing with RL presents a challenge due to the protracted convergence time required for the RL agent, especially within a constrained testing budget. To address this, we utilize a surrogate environment for the preliminary training of the RL agent. The surrogate environment used for RL training acts as a preliminary stage, streamlining the agent's learning process before it encounters the

main simulation. This environment simplifies certain complexities to allow for more rapid training phases. The details of the surrogate environment and the specific reward function utilized in our approach are further explained in the following subsections.

**3.2.1 Build Surrogate RL Training Environment.** The following changes were made to the full test environment to create the surrogate training environment:

- *Camera mode* in the UE4 [11] was used to simulate the flying and landing process of a UAV. The takeoff process occupies a substantial part of the simulation time. However, most violations occur due to in-flight marker detection and the landing process. The *Camera mode* excludes the takeoff process, yielding significant time savings.
- Objects move immediately to their respective states rather than displaying a smooth animation. During training, the goal is for the RL agent to learn trajectory adjustment based on the UAV and marker's position. Animation is not necessary in the training process.
- The marker is spawned in a fixed point and an object is spawned nearby the marker. Randomly spawning the marker and the object lead to a larger exploration space for RL, requiring more time to converge.
- Only one dynamic object is deployed in the surrogate environment, as once the single RL agent is converged. Multiple RL agents in the full simulation can load the weight to control multiple dynamic objects.

The DQN algorithm is used [43] as it is specifically designed for discrete action spaces. The surrogate training environment is built using UE4 [11] and AirSim [36]. The state of our RL input is a 4-dimensional vector that represents relative positional information between the object, the marker, and the UAV:

$$S = \{P_{obj,x} - P_{marker,x}, P_{obj,y} - P_{marker,y}, P_{uav,x} - P_{marker,x}, P_{uav,y} - P_{marker,y}\} \quad (3)$$

Where  $P_{obj}$  and  $P_{uav}$  are the positions of the object and the UAV respectively, and  $P_{marker}$  is the position of the marker.

Given the high degree of freedom within the UE4 [11] and AirSim [36] environments, we discretize the object's action space into categories of movement. The dimension of the discretized action space would be 5:

$$A = \{U, D, L, R, S\} \quad (4)$$

Where  $U, D, L, R, S$  represents moving up, down, left, right, and stationary. The RL agent will choose one action from the whole action space at each time step.

**3.2.2 Define Reward Function.** Our objective is to enable the object under control to maximize the percentage of UAV camera vision that is blocked. Therefore, we collect the semantic segmentation map from the UAV's camera which contains the semantic information for the marker and the object. Then, we calculate the percentage of the UAV camera's field of view being obscured. We define our reward function at each time step as follows:

$$R = \begin{cases} \frac{S_{gt}}{S_d}, & \text{if } S_d \neq 0 \\ 0, & \text{if } S_d = S_{gt} \text{ or } S_d = 0 \end{cases} \quad (5)$$

Where  $S_{gt}$  is the area of the marker, a ground truth retrieved from the simulator and  $S_d$  is the detected area of the marker through the camera at each time step. If  $S_{gt} = S_d$ , this signifies that the marker is not occupied and results in no reward. Similarly, if  $S_d = 0$ , it indicates that no marker has been detected which also results in no reward. To allow the RL to further benefit the GA, the

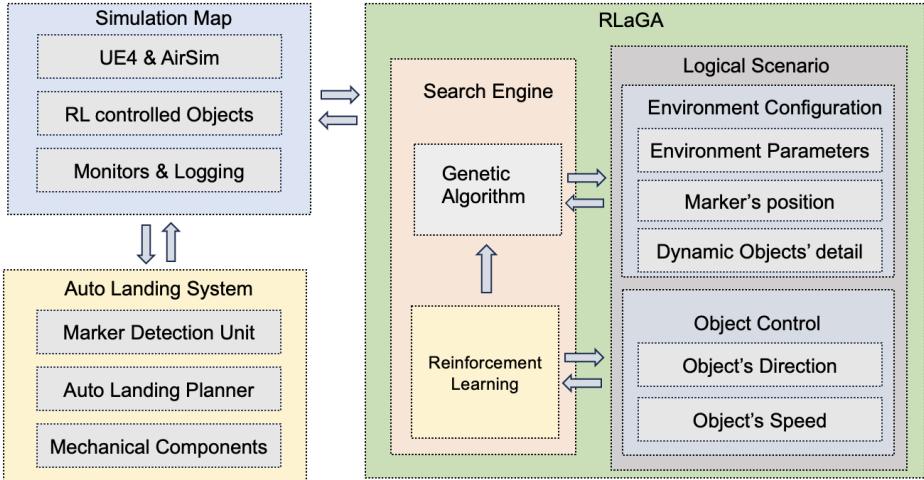


Fig. 5. Testing System Architecture

fitness function for RL augmented GA is updated as follows:

$$\text{FitnessValue} = \begin{cases} DTL + TTL + \frac{1}{\sum R}, & \text{if } \sum R \neq 0 \\ DTL + TTL, & \text{if } \sum R = 0 \end{cases} \quad (6)$$

Where  $\sum R$  is the accumulated reward value. A lower accumulated reward indicates a lower camera view obstruction throughout this simulation run, which would could result in UAV landing failure.  $\frac{1}{\sum R}$  is used to ensure that cases with a very low accumulated reward result in a high fitness value.

## 4 EXPERIMENT

### 4.1 Research Questions

The following research questions (RQs) were assessed to evaluate the performance of *RLaGA*:

- RQ1: How effective do the *RLaGA* generated test scenarios evaluate marker-based landing systems?
- RQ2: How effective is *RLaGA* at exposing landing violations compared to existing state-of-the-art techniques?
- RQ3: Can landing violations found in the simulation be verified in the real-world?

### 4.2 Experiment Setup

Figure 5 illustrates the overall architecture of our testing system, consisting of three main components. The simulation environment contains the environmental map or scenario to be simulated as well as the dynamic objects within that scenario, and the auto landing system consists of the software to be tested, which can include machine vision marker detection, path planning, and obstacle avoidance. The *RLaGA* algorithm controls the simulation parameters and searches for corner test cases that cause landing violations.

**4.2.1 Simulation Environment.** Simulated experiments are conducted using AirSim [36], an open-source drone simulation platform developed by Microsoft that is widely used in drone research [19, 39, 40]. It was chosen for its ability to render high-fidelity scenes and simulate realistic physical



Fig. 6. Example of test map in real-world (First row) and AirSim (Second row).

effects. In addition, it provides a rich API to control the movement of the drone, the positions of static objects, and the weather effects (e.g., rainy and snowy) of the simulated environment. It is noted, however, that the official AirSim package only provides a limited number of demonstration environments and features and does not provide APIs to add or control dynamic objects.

To achieve the simulation fidelity required to meet the research questions, customized simulation maps, marker assets, and new control APIs were added to AirSim. Two simulation maps, named Ground and Lawn, were created to replicate the real-world landing sites used by our industry partner, as shown in Figure 6. The Ground map is a city-like setting with mostly man-made structures such as concrete landing areas while the Lawn map is a primarily natural setting. ArUco markers [13], which are currently used as drone landing targets, were imported into simulation maps. AirSim APIs were used to position the ArUco markers at the start of simulation allowing the flight plan to be varied. Assets and corresponding animations of several dynamic objects, including *persons*, *birds*, *dogs* were also added to the simulation. New AirSim APIs were developed to allow the movement of these dynamic objects to be controlled.

**4.2.2 Marker-based Landing Systems.** The typical landing task for a marker-based landing system is defined as follows:

- (1) The drone is commanded to move to a specified GPS coordinate representing the approximate position of the target landing marker.
- (2) The landing system constantly attempts to detect the target marker during the transit flight.

- (3) Once the accurate position of the marker is identified, the transit is stopped and the landing procedure starts.
- (4) The landing system controls the landing descent such that the drone performs a landing precisely on the marker.

Three marker-based landing systems were implemented and integrated into the simulated drone and tested in the created simulation environments:

- *OpenCV-MLS*: a marker detection module, provided by our industry partner, that utilizes traditional computer vision algorithms [6] to detect and estimate the marker’s position.
- *TPHYoLo-MLS*: an implementation of the TPHYoLo [51] deep learning model as the marker detection module. This model adopts the transformer mechanism [44] to improve performance when detecting small-size objects and is a proven method for use in drone-related research [51].
- *MM-MLS*: a multi-module structured landing system that utilizes TPHYoLo to detect the position of the marker and a planning module, Ego-planner [50], to plan a movement trajectory for the drone to reach the identified marker based on obstacle information obtained from depth images. We chose to use Ego-planner in the landing system because it is the current state-of-the-art open-sourced planner that can achieve real-time obstacle detection and trajectory planning.

Though some other marker-based landing systems have been proposed [7, 25], their models were not available as open-source at the time of writing and, thus, were not evaluated in this study.

### 4.3 Experiment Design

**4.3.1 RQ1.** The ability of the *RLaGA* method to generate diverse test scenarios that cause failed landings is evaluated. We assessed *RLaGA* on three landing systems *OpenCV-MLS*, *TPHYoLo-MLS*, and *MM-MLS* in two simulation maps *Ground* and *Lawn* respectively. Each landing system was assessed by using *RLaGA* generating 300 test scenarios (our test budget) in total (20 seeds in each generation, and running 15 generations in total). Data for each scenario run was saved, including the scenario seed, a video recording, the drone trajectory data, and any drone-object collision events.

The simulation data was used to identify whether a test scenario caused a landing violation of the tested landing system. A landing violation was said to occur either when the drone collides with other objects, or when the final landing position of the drone is outside the bounds of the landing marker (1.5 meters from the marker center in this study). A *landing violation percentage* was calculated as the ratio of scenarios with violations divided by the total number of scenarios. This value measures how effective the *RLaGA* method is at finding landing violation scenarios.

The saved landing recordings were also analyzed to identify the *number of unique violation causes*, which qualitatively measures the diversity of generated test scenarios. The *violation cause* is the reason for the failed landing. For example, the reason could be that the marker is not detected due to the environmental weather, the static objects, or the dynamic objects. A large number of unique violation causes demonstrates the ability of the *RLaGA* method to generate diverse landing violation scenarios.

**4.3.2 RQ2.** We compared the performance of *RLaGA* with two baseline methods on *MM-MLS* in two simulation maps. The first baseline method is *Random*, where all scenario seeds in all generations are randomly sampled, and the second baseline method is *Many-Objective GA*, where the fitness function is set as Equation 1 [2, 32, 42]. In addition, to evaluate the effectiveness of the RL component in the proposed method, we conducted an ablation study by only using the GA (*Diverse GA*) component in the proposed method as another baseline to compare.

	<i>OpenCV-MLS</i>	<i>TPHYoLo-MLS</i>	<i>MM-MLS</i>
Map Ground	69.47%	47.10%	55.42%
Map Lawn	57.37%	27.68%	36.84%

Table 1. Landing violation percentage for different landing systems across two maps

The efficiency metric is measured as **Top-K** [10], which evaluates how fast the first  $K$  test scenarios cases that cause landing violations are generated. In this paper, the  $K$  is set as 10. The *landing violation percentage* metric, from RQ1, is used to evaluate how many of the generated scenarios produce landing violations in this unit of time. Diversity was measured with two metrics: *parameter distance* and *3-D trajectory coverage*.

**Parameter distance** was derived from the idea of novel search [22, 23]. A higher parameter distance means the generated violation seeds have a higher diversity. Parameter distance is calculated for all scenario seeds of all landing violation scenarios as shown in Formula 7:

$$\rho(x) = \frac{1}{n} \sum_{j=1}^n d(x, x_j) \quad (7)$$

Where  $x$  is the property of chromosome in one generation,  $d$  is a function of Euclidean distance, and  $n$  is the number of found violation cases.

**3D-Trajectory Coverage** is a trajectory coverage metric that measures how violated trajectories cover the simulation map, and is adapted from an autonomous driving system implementation [17]. This study implements the metric in 3D in order to measure the trajectory coverage during a 3D marker-based UAV landing task. The simulation environment is divided into several  $2 \times 2 \times 2$  meter cells. A 3-dimensional array is used to index each cell in the grid. All entries in the array are initialized to *false*. The value of an entry is set *true* if the corresponding grid cell is visited once by the UAV during the test.

**4.3.3 RQ3.** Various real-world scenarios were reproduced based on landing violation scenarios that were found in the simulation. The *MM-MLS* landing system was deployed on a custom-built drone to achieve this. The drone features a 295mm, frame size, 5-inch 3-blade propellers, 1750KV motors, an Intel D455 camera, and an NVIDIA Jetson Nano onboard computer. The marker detection module in *MM-MLS* is deployed on the drone to identify markers in real time. The software system uses the YOLO marker detection and is configured to initiate the autoland software with primitive obstacle avoidance using range finders when the 0.70 confidence threshold of marker detection is exceeded. The confidence threshold has been predefined as per the camera's detection range in the experimental setup. The drone is programmed to detect a specific physical marker ("markerID:3") and commence landing after 5 seconds of continuous marker detection above the confidence threshold. The reproduced real-world scenarios primarily featured landing violations caused by incorrect ID markers, weather conditions such as glare/fog, and a dynamic or static object obstruction.

## 5 RESULTS

### 5.1 Finding Diverse Violation Types in Different AutoLand Systems and Maps (RQ1)

Table 1 showcases the test results of our method across various iterations of the landing system, which evolved over three versions, and different test maps. It highlights its effectiveness in identifying landing violations for these emerging versions of industry-standard UAV landing systems. As the landing system advances, a noticeable reduction in generated violations is observed, especially

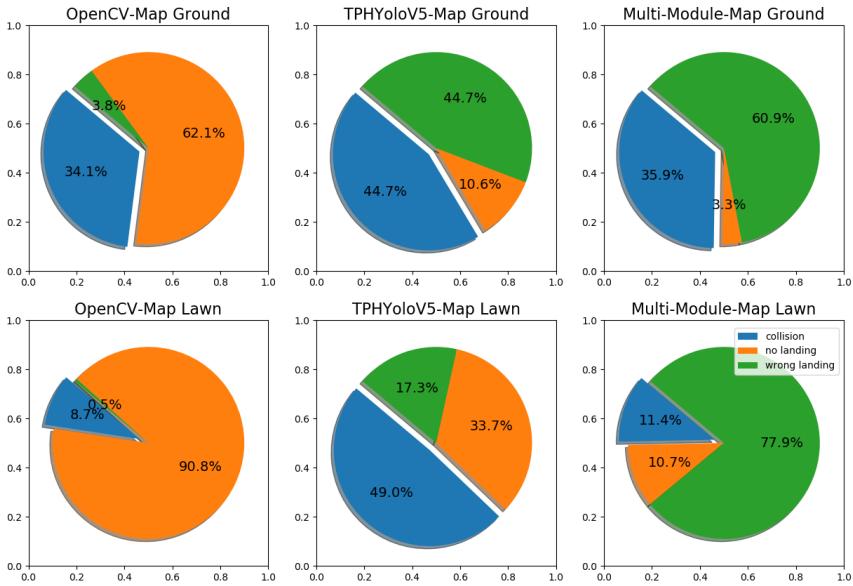


Fig. 7. Visualized violation cases found in various scenarios.

transitioning from *OpenCV-MLS* to *TPHYolo-MLS*, where the violation rate plunged by up to 29.69%. This decline emphasizes the augmented robustness of deep learning models compared to OpenCV models for marker detection. Interestingly, the landing violation percentages on *MM-MLS* in two maps are even higher than on *TPHYoLo-MLS*, even though *MM-MLS* has a planner module with the obstacle avoidance functionality. The reason is that the planner itself has difficulties in planning correct trajectories and causes landing violations in some cases, which is analyzed in Section 5.1.3.

By analyzing the saved simulation data and test recordings, we manually categorize our generated landing violations into three different types and indicate some examples for each violation type.

**5.1.1 Type I.** The first violation type is landing due to false positive detection. Figure 8 illustrates two example sequences of landing on a false positive location. At the top example sequence, initially, likely due to ground reflection, the ground is mistakenly detected as a marker with high confidence, prompting the UAV to commence landing. Despite losing the marker's position in subsequent attempts, the UAV continues to land, resulting in a landing at a false positive location. In the bottom example sequence, the white marker is initially recognized as the landing marker. As the landing process unfolds, even though the confidence fluctuates, the UAV still ultimately decided to land in a false positive position.

**5.1.2 Type II.** The second type of violation is the failure to land because no object is detected as a marker. Figure 9 showcases several instances where no marker is detected. In the first provided example, a white static object partially covers the marker, and the day is foggy. These conditions result in the marker not being detected. In the second example, the marker is placed near a chair, and the presence of a nearby black object prevents the marker from being detected. In the third



Fig. 8. Examples of Violation Type I: UAV lands on a false positive location due to wrong detection



Fig. 9. Examples of Violation Type II: No marker is detected

example, extreme weather conditions cause the marker to go undetected, or the confidence is significantly low. Consequently, no landing occurs.

**5.1.3 Type III.** The third violation type occurs when the obstacle avoidance system doesn't have sufficient reaction time to evade sudden dynamic objects. In the first row of Figure 10, a person abruptly appears in the UAV's camera view. Initially, this person is at a distance that does not impact the UAV's planner. However, as the person moves quickly closer to the marker, the UAV descends to a lower altitude, leaving insufficient time for the UAV's planner to devise a new path to avoid the person. A similar situation is depicted in the example in the second row, but this occurs on the lawn map and is caused by a dog and a person.

The third row in the Figure 10 illustrates the failure of the obstacle avoidance system to avoid obstacles. Initially, the marker is detected, and the UAV begins its landing sequence. However, branches surrounding the marker, situated only around the marker's border, do not trigger the obstacle avoidance system, leading to a collision.

In Figure 7, the distribution of various violation types across different maps and systems is illustrated. It is evident that for the OpenCV MLS system, the majority of violations are categorized under 'no landing.' This is primarily attributable to the limited robustness of the OpenCV model as a detection algorithm. Various factors, similar to those described in Type II, can easily disrupt its functionality, resulting in an inability to detect landing markers and consequently, no landing occurs.

In contrast, for the TPHYoloV5 MLS and MM MLS systems, we observe different patterns. Those two systems are using deep learning-based marker detection modules, which exhibits high robustness. Meanwhile, the MM MLS system is further equipped with an obstacle avoidance module, reflecting in a discernible reduction in the occurrence of collision-related violations compared to the TPHYoloV5 MLS system. This indicates the contribution of the obstacle avoidance module in



Fig. 10. Examples of Violation Type III: Collisions due to obstacle avoidance failures

Method	Map	Landing violation percentage	Top-10	Rounds to cover all types	Parameter distance	3D trajectory coverage
RLaGA		<b>50.60%</b>	<b>19</b>	<b>64</b>	<b>62.47</b>	20.89
Multi-Obj GA	Ground	28.41%	48	Not Found	27.40	11.00
		41.36%	25	Not Found	24.83	<b>31.57</b>
		16.07%	55	132	45.74	27.22
RLaGA		35.26%	36	<b>70</b>	<b>70.53</b>	23.1
Multi-Obj GA	Lawn	15.39%	55	Not Found	36.67	10.18
		30.24%	41	Not Found	25.36	28.42
		<b>39.58%</b>	<b>32</b>	Not Found	52.14	<b>40.86</b>

Table 2. Quantitative result of violation case generating efficiency and diversity for different methods, best is marker in bold.

enhancing the safety and reliability of the MM MLS system in diverse and dynamic environments. Interestingly, our method successfully detects a significant percentage of incorrect landings for MM MLS system. Upon a detailed review of the reports, we determined that these discrepancies are primarily due to the stability issues associated with Ego-planner we employed. Despite Ego-planner being among the most advanced and quickest available for drones [48], our system pinpointed its specific challenge: when the drone flies close to obstacles from the air to the ground, the planner often gets stuck or produces incorrect trajectories that lead to potential collisions.

In summary, our method’s effectiveness is showcased by its capability to identify all types of violations across different systems. Leveraging these cataloged violation types, we will proceed to compare our method with other baselines in RQ2.

## 5.2 Comparison to Baselines (RQ2)

We compared our method with baselines and ablation methods based on our analysis in RQ1. As shown in Table 2, in the Ground map, we find *RLaGA* was able to find more violations with 50.60% which is up to 22.19% better than the state-of-the-art methods in 300 runs. *RLaGA* was also able to find all violation types in 64 rounds while the other baselines were either unable to discover all types or took more rounds to find this diversity (132 rounds for Diverse GA). In terms of the

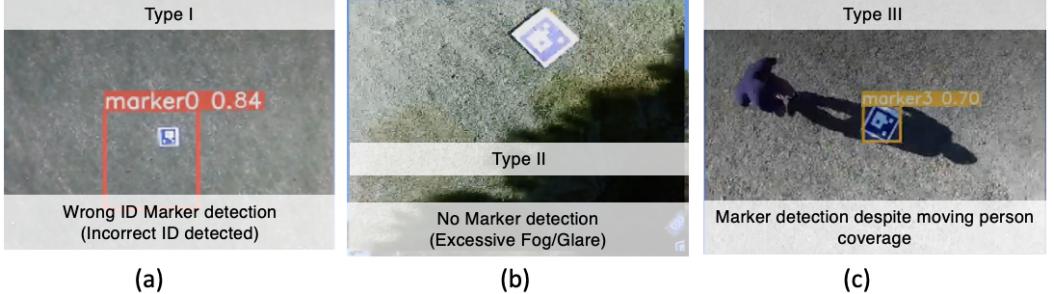


Fig. 11. Example of our real-world violation reproduction, (a) a false positive detection case, (b) failure to land because no object is detected as a marker, and (c) is a violation due to obstruction that highlights the detected marker after perturbation leading to a potential collision.

evaluation metrics for diversity, *RLaGA* was the best with 62.47 in Parameter distance. Even though Random performs best in 3D trajectory coverage but those trajectories are merely random and couldn't result in violation detection as shown in other metrics. In terms of efficiency of finding bugs, *RLaGA* performs best with 19 test cases to reveal the top 10 bugs which is up to 189.47% better than the other baselines. Similarly statistics can be shown in the Lawn map though Diverse GA surprisingly outperforms *RLaGA* in violation percentage and top-10, but the improvement is marginal. Considering its performance in the ground map, Diverse GA is not robust and only *RLaGA* can uncover all types of violation which is crucial.

### 5.3 Real-World Reproduction (RQ3)

We conducted real-world tests to confirm whether the violation cases found by the *RLaGA* algorithm through simulation appear in practice. Figure 11 shows the different failure events encountered during actual drone flight and landing due to violations in detecting the safe landing spot indicated by the marker. In our real-world test cases, we explored different weather conditions, lighting settings, and different obstructions, such as a static test with an obstructing tree branch, and a dynamic test with a moving person. In Fig 11 (a), we observe that despite the desired marker ID being set to 3, the drone detects a different ID 0 as the marker, creating a false positive case, which is an example of a Type I violation. Fig 11 (b) demonstrates the Type II test case where an excessive glare makes the marker unrecognizable thereby incorrectly preventing the AutoLand process from being initiated. Lastly, Fig 11 (c) demonstrates a Type III violation by including a moving person near the marker that doesn't allow sufficient time for the basic obstacle avoidance model to detect the obstruction, continuing the landing process that may lead to a collision. These findings demonstrate that the errors identified by our method can be replicated in real-world settings.

## 6 THREAT TO VALIDITY

The main threat to external validity is regarding the generalization of the proposed method. To address the problem, we have conducted experiments to generalize our search method on different landing systems including *OpenCV-MLS*, *TPHYoLo-MLS*, and *MM-MLS*. In addition, we tested these landing systems on different maps based on real-world deployment sites of our industry partners. Our method has demonstrated the capability to identify diverse corner cases across these different automatic landing systems and maps. A potential threat to internal validity is the fidelity of the simulation environment. To address this concern, for each deployment map, we incorporated dynamic object types found in the actual corresponding deployment sites into Airsim. Additionally,

we ensured our maps closely resembled the real deployment environments. The positive outcomes from our real-world tests further alleviate this validity concern. The threat to the construct validity is the adaptation of the baseline test generation methods. As there is little related work on testing marker-based landing systems, it is difficult to find other methods we can directly use as baselines to compare. To solve the problem, we adapted the state-of-the-art search-based test generation methods in the ADS testing community. We kept their main ideas and adapted their fitness functions to fit the specific requirements of a marker-based UAV landing task.

## 7 CONCLUSION

In this paper, we introduce a unique approach to tackle the challenge of producing numerous genuine and varied violation cases for marker-based UAV landing scenarios. Our innovative search method incorporates a groundbreaking GA module with a pioneering chromosome seed representation. It generates diverse environment seeds offline through nuclear gene-level mutations and property-level crossovers. Additionally, an RL module is employed for online testing, enabling the manipulation of online object trajectories within these settings. This strategy effectively overcomes the generation hurdle, supported by experimental results that validate its capability to detect a range of error types replicable in real-world contexts.

This study also lays the groundwork for exciting future research avenues. Firstly, the effectiveness of our method can be further evaluated using simulation maps that reflect diverse real-world sites. Secondly, we elucidate the integration of offline and online testing by leveraging novel chromosome representations of test seeds combined with a surrogate-trained reinforcement learning module. Furthermore, while our current model utilizes a conventional RL algorithm to oversee dynamic object movements in test situations, delving into multi-agent RL could empower objects to adapt collaboratively and showcase unique behaviors. Such an exploration might reveal an expanded spectrum of test situations and edge cases.

## REFERENCES

- [1] Open source for ardupilot open source autopilot, <https://github.com/ArduPilot/ardupilot>.
- [2] Abdessalem, R.B., Panichella, A., Nejati, S., Briand, L.C., Stifter, T.: Testing autonomous cars for feature interaction failures using many-objective search. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 143–154 (2018)
- [3] Administration, F.A.: (2022), <https://www.faa.gov/uas>
- [4] Authority, C.A.S.: Drone rules (2022), <https://www.casa.gov.au/knowyourdrone/drone-rules>
- [5] Baca, T., Stepan, P., Spurny, V., Hert, D., Penicka, R., Saska, M., Thomas, J., Loianno, G., Kumar, V.: Autonomous landing on a moving vehicle with an unmanned aerial vehicle. *Journal of Field Robotics* **36**(5), 874–891 (2019)
- [6] Bradski, G.: The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000)
- [7] Brunner, G., Szebedy, B., Tanner, S., Wattenhofer, R.: The urban last mile problem: Autonomous drone delivery to your balcony. In: 2019 international conference on unmanned aircraft systems (icuas). pp. 1005–1012. IEEE (2019)
- [8] Chen, L., Yuan, X., Xiao, Y., Zhang, Y., Zhu, J.: Robust autonomous landing of uav in non-cooperative environments based on dynamic time camera-lidar fusion (2020)
- [9] Corso, A., Du, P., Driggs-Campbell, K., Kochenderfer, M.J.: Adaptive stress testing with reward augmentation for autonomous vehicle validation. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC). pp. 163–168. IEEE (2019)
- [10] Deng, Y., Zheng, X., Zhang, M., Lou, G., Zhang, T.: Scenario-based test reduction and prioritization for multi-module autonomous driving systems. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 82–93 (2022)
- [11] Epic Games: Unreal engine, <https://www.unrealengine.com>, accessed 2019-04-25
- [12] Esnaashari, M., Damia, A.H.: Automation of software test data generation using genetic algorithm and reinforcement learning. *Expert Systems with Applications* **183**, 115446 (2021)
- [13] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F.J., Marín-Jiménez, M.J.: Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition* **47**(6), 2280–2292 (2014)
- [14] Gupta, D., Ghafir, S.: An overview of methods maintaining diversity in genetic algorithms. *International journal of emerging technology and advanced engineering* **2**(5), 56–60 (2012)
- [15] Haq, F.U., Shin, D., Briand, L.: Many-objective reinforcement learning for online testing of dnn-enabled systems. *arXiv preprint arXiv:2210.15432* (2022)
- [16] Haq, F.U., Shin, D., Briand, L.C., Stifter, T., Wang, J.: Automatic test suite generation for key-points detection dnns using many-objective search. *arXiv preprint arXiv:2012.06511* (2020)
- [17] Hu, Z., Guo, S., Zhong, Z., Li, K.: Coverage-based scene fuzzing for virtual autonomous driving testing. *arXiv preprint arXiv:2106.00873* (2021)
- [18] Koren, M., Alsaif, S., Lee, R., Kochenderfer, M.J.: Adaptive stress testing for autonomous vehicles. In: 2018 IEEE Intelligent Vehicles Symposium (IV). pp. 1–7. IEEE (2018)
- [19] Lai, K.T., Chung, Y.T., Su, J.J., Lai, C.H., Huang, Y.H.: Ai wings: an aiot drone system for commanding ardupilot uavs. *IEEE Systems Journal* (2022)
- [20] Lambora, A., Gupta, K., Chopra, K.: Genetic algorithm-a literature review. In: 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon). pp. 380–384. IEEE (2019)
- [21] Lee, B., Saj, V., Kalathil, D., Benedict, M.: Intelligent vision-based autonomous ship landing of vtol uavs. *Journal of the American Helicopter Society* **68**(2), 113–126 (2023)
- [22] Lehman, J., Stanley, K.O.: Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation* **19**(2), 189–223 (2011)
- [23] Lehman, J., Stanley, K.O., et al.: Exploiting open-endedness to solve problems through the search for novelty. In: ALIFE. pp. 329–336 (2008)
- [24] Li, G., Li, Y., Jha, S., Tsai, T., Sullivan, M., Hari, S.K.S., Kalbarczyk, Z., Iyer, R.: Av-fuzzer: Finding safety violations in autonomous driving systems. In: 2020 IEEE 31st international symposium on software reliability engineering (ISSRE). pp. 25–36. IEEE (2020)
- [25] Lin, S., Jin, L., Chen, Z.: Real-time monocular vision system for uav autonomous landing in outdoor low-illumination environments. *Sensors* **21**(18), 6226 (2021)
- [26] Lou, G., Deng, Y., Zheng, X., Zhang, M., Zhang, T.: Testing of autonomous driving systems: where are we and where should we go? In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 31–43 (2022)
- [27] Lu, C.: Test scenario generation for autonomous driving systems with reinforcement learning. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). pp. 317–319. IEEE (2023)

- [28] Lu, C., Shi, Y., Zhang, H., Zhang, M., Wang, T., Yue, T., Ali, S.: Learning configurations of operating environment of autonomous vehicles to maximize their collisions. *IEEE Transactions on Software Engineering* **49**(1), 384–402 (2022)
- [29] Lu, Y., Shao, K., Sun, W., Sun, M.: Rgchaser: A rl-guided fuzz and mutation testing framework for deep learning systems. In: 2022 9th International Conference on Dependable Systems and Their Applications (DSA). pp. 12–23. IEEE (2022)
- [30] Marcu, A., Costea, D., Licaret, V., Pirvu, M., Slusanschi, E., Leordeanu, M.: Safeuav: Learning to estimate depth and safe landing areas for uavs from synthetic data. In: Proceedings of the European Conference on Computer Vision (ECCV) Workshops. pp. 0–0 (2018)
- [31] Mittal, M., Mohan, R., Burgard, W., Valada, A.: Vision-based autonomous uav navigation and landing for urban search and rescue. In: The International Symposium of Robotics Research. pp. 575–592. Springer (2019)
- [32] Panichella, A., Kifetew, F.M., Tonella, P.: Reformulating branch coverage as a many-objective optimization problem. In: 2015 IEEE 8th international conference on software testing, verification and validation (ICST). pp. 1–10. IEEE (2015)
- [33] Reddy, S., Lemieux, C., Padhye, R., Sen, K.: Quickly generating diverse valid test inputs with reinforcement learning. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. pp. 1410–1421 (2020)
- [34] Schmidt, T., Hauer, F., Pretschner, A.: Understanding safety for unmanned aerial vehicles in urban environments. In: 2021 IEEE Intelligent Vehicles Symposium (IV). pp. 638–643. IEEE (2021)
- [35] Schmidt, T., Pretschner, A.: Stellauav: A tool for testing the safe behavior of uavs with scenario-based testing (tools and artifact track). In: 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE). pp. 37–48. IEEE (2022)
- [36] Shah, S., Dey, D., Lovett, C., Kapoor, A.: Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: Field and Service Robotics (2017), <https://arxiv.org/abs/1705.05065>
- [37] Shakhatreh, H., Sawalmeh, A.H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., Othman, N.S., Khreichah, A., Guizani, M.: Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *Ieee Access* **7**, 48572–48634 (2019)
- [38] Sharif, A., Marijan, D.: Adversarial deep reinforcement learning for trustworthy autonomous driving policies. arXiv preprint arXiv:2112.11937 (2021)
- [39] Shimada, T., Nishikawa, H., Kong, X., Tomiyama, H.: Pix2pix-based monocular depth estimation for drones with optical flow on airsim. *Sensors* **22**(6), 2097 (2022)
- [40] Song, Y., Steinweg, M., Kaufmann, E., Scaramuzza, D.: Autonomous drone racing with deep reinforcement learning. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1205–1212. IEEE (2021)
- [41] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
- [42] Tian, H., Jiang, Y., Wu, G., Yan, J., Wei, J., Chen, W., Li, S., Ye, D.: Mosat: finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 94–106 (2022)
- [43] Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence. vol. 30 (2016)
- [44] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. *Advances in neural information processing systems* **30** (2017)
- [45] Vetrella, A.R., Fasano, G., Renga, A., Accardo, D.: Cooperative uav navigation based on distributed multi-antenna gnss, vision, and mems sensors. In: 2015 International Conference on Unmanned Aircraft Systems (ICUAS). pp. 1128–1137. IEEE (2015)
- [46] Vincent, J.: Food delivery drone lands on power lines resulting in power outage for thousands. ABC News (2022-09-30), <https://www.theverge.com/2022/9/30/23380044/food-delivery-drone-knocks-out-power-australia-wing>
- [47] Wegener, J., Bühler, O.: Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In: Genetic and Evolutionary Computation—GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26–30, 2004. Proceedings, Part II. pp. 1400–1412. Springer Berlin Heidelberg (2004)
- [48] Yu, H., de Croon, G.C.E., De Wagter, C.: Avoidbench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors. In: 2023 IEEE International Conference on Robotics and Automation (ICRA). pp. 9183–9189. IEEE (2023)
- [49] Zhong, Z., Kaiser, G., Ray, B.: Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Transactions on Software Engineering* (2022)
- [50] Zhou, X., Wang, Z., Ye, H., Xu, C., Gao, F.: Ego-planner: An esdf-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters* **6**(2), 478–485 (2020)
- [51] Zhu, X., Lyu, S., Wang, X., Zhao, Q.: Tph-yolov5: Improved yolov5 based on transformer prediction head for object detection on drone-captured scenarios. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 2778–2788 (2021)