Evolutionary Single Shot Tokenized GAN for Simulation Environment Generation and Fault Detection

Charbel Bou Maroun
R&I department
InMind .ai
Beirut, Lebanon
charbel.boumaroun@inmind.ai

George Daou
E.C.E. department
Lebanese American University
Byblos, Lebanon
georges.daou@lau.edu

Charbel Aoun[1]
Institut Catholique d'Arts et Métiers
ICAM, School of Engineering
Toulouse, France
charbel.aoun@icam.fr

Joe Tekli ✉
E.C.E. department
Lebanese American University
Byblos, Lebanon
joe.tekli@lau.edu.lb

***Abstract*—Advanced robotic technologies such as self-driving cars, autonomous robots, and unmanned aerial vehicles require rigorous testing to expose flaws in their control software. Developing suitable testing environments is a challenging task due to the difficulty of designing adequate testing environments that uncover faults in the robotics control software. In this paper, we create a custom Simulation Evolutionary Single-Shot Generative Adversarial Network titled SESS-GAN. It makes use of single shot tokenized GANs to generate a preliminary simulation environment model, which is then fed into an evolutionary algorithm for fine-tuning. This process increases the risk score of the simulations; meaning these environments will cause the autonomous robot control software to fail/near fail, allowing testers to evaluate its performance accordingly. Empirical results highlight the potential of our solution with multiple use case environments.**

Keywords—Autonomous driving robots, testing environment, simulation, generative adversarial network, evolutionary algorithm.

I. Introduction

There is an ever-increasing demand for advanced robotic technologies such as self-driving cars, independent robots, and unmanned aerial vehicles. These autonomous systems have the potential to revolutionize the way we live and work. As a result, it becomes increasingly crucial to ensure that their control software is thoroughly tested to identify and mitigate any potential flaws. Therefore, creating suitable testing environments for these advanced robotic technologies becomes a major requirement. Nonetheless, designing environments that can uncover faults in autonomous robotics software is a difficult and challenging task given the complexity of these systems.

In this study, we aim to develop a model which generates testing environments that can simulate real-world scenarios and identify flaws in their control software. To do so, we put forward a custom Generative Adversarial Network (GAN) named SESS-GAN (Simulation Evolutionary Single-Shot GAN). It leverages single-shot tokenized GANs to generate a preliminary model that is then fed into an evolutionary algorithm for fine-tuning. Through this process, we can develop simulations that increase the risk score of their environments, which means that they will cause the autonomous robot control software to fail or near-fail more frequently. Our work can have significant implications for the development and testing of advanced robotic technologies, by creating simulations that are more effective at identifying potential flaws in control software, ultimately leading to safer and more reliable autonomous systems.

Section II briefly describes the related works. Section III described our approach. Section IV presents the evaluation metrics and results, before concluding in Section V.

II. Related Works

A. Procedural Content Generation

Several procedural content generation techniques allow to create simulation environments to test and simulated robotic control software [1]. Procedural generation has been widely used in the gaming industry to create realistic and immersive environments. It is based on algorithms that generate content automatically, allowing developers to create vast and diverse game worlds without the need for manual creation of every asset. The advantage of procedural generation is that it can create environments that are highly realistic, with complex and nuanced rules that accurately reflect the real-world conditions that autonomous robots will encounter [7]. By generating environments with a wide range of variables, such as terrain, weather, and lighting, it is possible to test how control software performs under a range of conditions. This can provide valuable insight into the behavior of autonomous systems and help to identify and mitigate potential flaws.

However, defining complex rules can be difficult with procedural generation, especially in situations where the rules are not well-defined or where the environment is particularly complex. In some cases, it may be necessary to manually define rules for each new environment, which can be extremely time-consuming and sometimes impossible. In addition, there is always the risk that the generated environment may not accurately reflect the real-world conditions that autonomous robots will encounter. To overcome these challenges, researchers are exploring new techniques for procedural generation that can automatically learn the rules of an environment through machine learning algorithms [5]. By training these algorithms on real-world data, it may be possible to generate environments that accurately reflect the real-world conditions that autonomous robots will encounter, without the need for manual rule definition [13].

Procedural generation has the potential to be an effective technique for creating simulation environments to test robotic control software. Yet, the challenges associated with defining complex rules and generating environments that accurately reflect real-world conditions must be carefully considered and addressed to ensure the effectiveness of this approach.

B. GAN-based Generation

More recent solutions to perform content generation utilize machine learning models, namely Generative Adversarial Networks (GANs), which have been used successfully in various applications, including generating maps and environments [1, 3]. Unlike procedural generation, which requires manual rule definition, GANs can generate environments by learning from a dataset of real-world examples. This makes it possible to generate environments that accurately reflect the real-world conditions that autonomous robots will encounter [9, 15]. However, creating a proper dataset for GAN training can be a significant challenge. In some cases, it may be necessary to collect large amounts of data, which can be time-consuming and expensive. Additionally, the quality of the dataset can have a significant impact on the performance of the GAN. If the dataset is biased or incomplete, the generated environments may not accurately reflect the real-world conditions that autonomous robots will encounter. Another major limitation of regular GANs is that they require a differentiable loss function [2, 4]. This means that the loss function must be smooth and continuous, allowing gradients to be calculated and used for backpropagation during the training process. While this is not a problem for many applications, it can be a significant limitation for certain scenarios, such as in robotics, where it is difficult to differentiate the *finding faults in a control software* metric, since it will come from running an actual robotic simulation on the generated environments. To overcome some of these limitations, researchers are exploring new techniques for training GANs in non-differentiable environments [8]. One approach is to use reinforcement learning, where the GAN learns from its interactions with the environment rather than relying solely on a predefined dataset [6, 12].

979-8-3315-8747-5/25/$31.00 ©2025 IEEE

[1] C. Aoun is co-affiliated with the Lab-STICC, CNRS UMR, ENSTA (Ecole Nationale Supérieure de Techniques Avancées), Brest, France

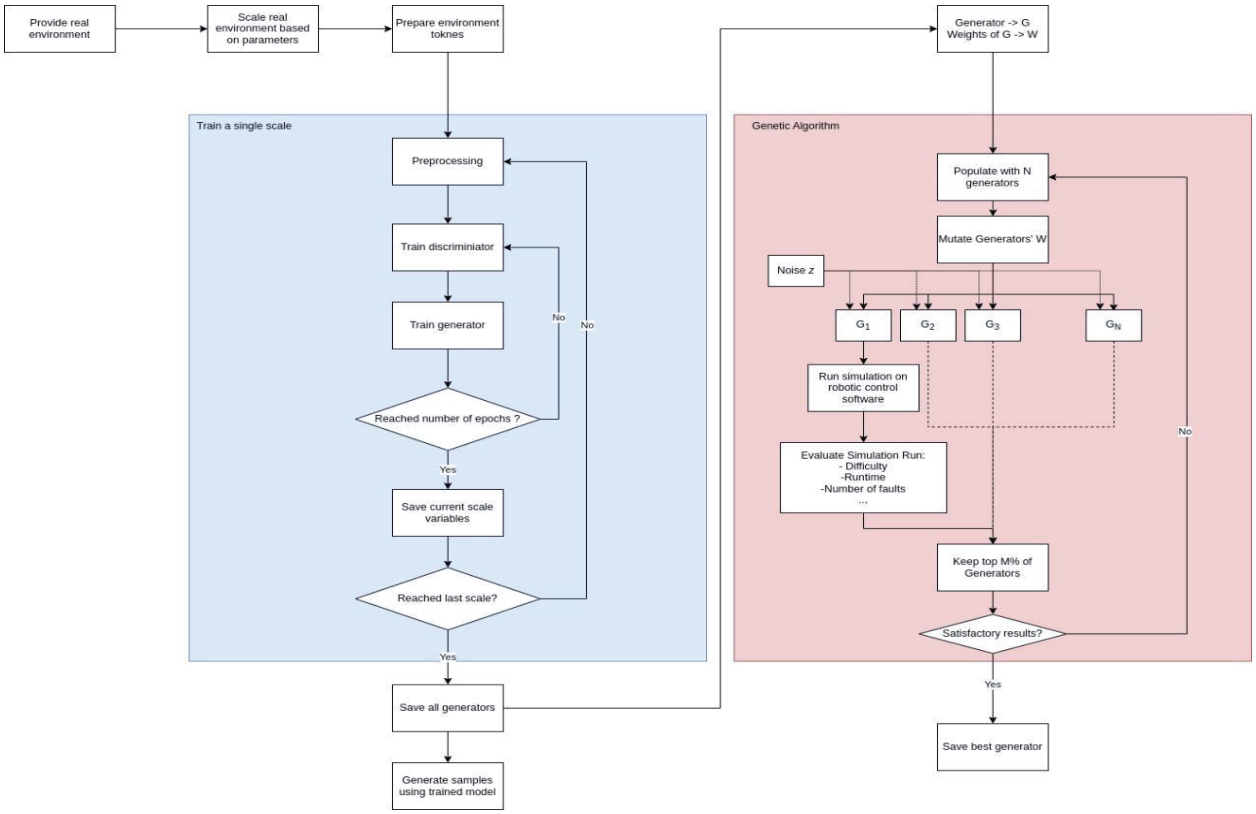


Fig. 1. Overall SESS-GAN architecture

In conclusion, GANs have the potential to be an effective technique for generating simulation environments to test robotic control software. However, creating a proper dataset and overcoming the limitations associated with differentiable loss functions are important challenges that must be addressed to ensure the effectiveness of this approach. Ongoing research in this area is expected to lead to new techniques and advancements that will make GANs an even more valuable tool for testing and improving autonomous systems.

C. Evolutionary Search-based Frameworks

Evolutionary search-based frameworks represent another approach to generating realistic simulation environments for testing robotic control software. This technique involves using optimization algorithms to iteratively refine an initial solution based on a fitness function. By continuously evaluating the fitness of the solution and making incremental improvements, this approach can generate simulation environments that are well-suited for testing the performance of autonomous systems. Evolutionary search-based frameworks have been successfully applied in a variety of applications, including generating maps for simulations [10].

However, this approach can be computationally intensive, requiring significant processing power and time to generate high-quality simulation environments [16]. Another challenge with this approach is defining appropriate fitness functions [11]. The fitness function represents the criteria that the optimization algorithm uses to evaluate the quality of the generated simulation environment. It is essential to define a fitness function that accurately captures the real-world conditions that the autonomous robots will encounter. However, determining the appropriate fitness function can be a difficult and time-consuming task, particularly for complex environments [10, 11].

III. Proposal

In this study, we put forward a custom model named Simulation Evolutionary Single-Shot GAN (SESS-GAN, cf. Fig. 1). It leverages single-shot tokenized GANs to generate a preliminary model that is then fed into an evolutionary algorithm for fine-tuning. Compared with legacy GANs, our model is designed to develop simulations that increase the risk score of their environments, which means that they will cause the autonomous robot control software to fail or near-fail more frequently.

A. Overall Process

We train a single-shot tokenized GAN on a single environment. The model is composed of a pyramid of fully convolutional GANs, with each GAN being responsible for learning the patch distribution at a different scale of the image. This allows the generation of new samples of arbitrary size and aspect ratio, with significant variability while preserving the visual content of the original image. Unlike previous single-image GAN schemes, single shot GAN is not limited to texture images and is not conditional, meaning that it generates samples from noise.

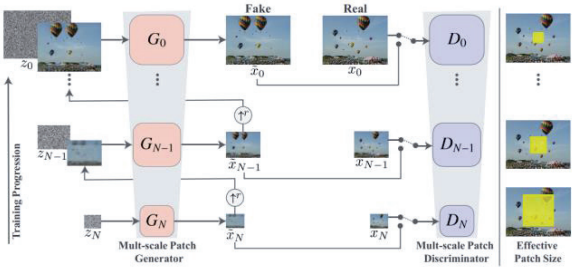


Fig. 2. Overall single shot GAN architecture (reported from [14])

This pre-trained GAN serves as the initial population for genetic evolution tuning. The genetic evolution tuning process involves creating a population of the pre-trained GAN and evaluating each member's fitness score, which reflects how well it performs on the environment generation task. The fitness score calculation does not require differentiability, making it applicable to environments where the loss metric is non-differentiable. In our case, it will be computed by running simulations on the generated environments of each member of the population and combining metrics which we will discuss in

the following sections (e.g., similarity of the generated environment to the original environment, the diversity of the generated environments, and the playability of the levels). These metrics are used to evaluate the fitness of each member and rank them accordingly. The best-performing GANs are selected and subjected to genetic operations such as crossover and mutation, creating a new population of GANs with slightly different weights and biases. This process is repeated for generations until the GANs converge to a satisfactory performance level.

The main advantage of our SESS-GAN solution is its ability to generate environments that are coherent and realistic while also being adaptable to different scenarios. The single-shot tokenized GAN learns the internal distribution of patches within the image and generates high-quality, diverse samples that carry the same visual content as the original image. The genetic evolution tuning process helps to fine-tune the GAN's parameters and biases to better fit the environment generation task, ensuring that the generated environments are more diverse and accurate.

Furthermore, SESS-GAN can generate environments that are adaptable to different scenarios, as it can be fine-tuned for different environments by modifying the fitness function. This flexibility allows our solution to be applied to a wide range of applications beyond video game environment generation, such as virtual reality, computer graphics, and simulation. Overall, SESS-GAN is a powerful and flexible solution for generating coherent and realistic environments.

Fig. 3 shows an example showing a simulation robot control software, and Fig. 4 shows the corresponding Unity simulation a generated factory environment.

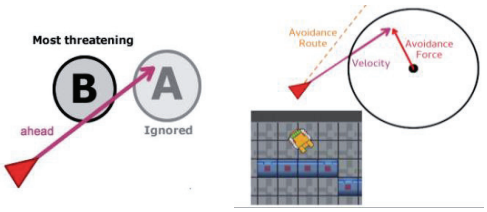


Fig. 3: Nearness-Diagram algorithm

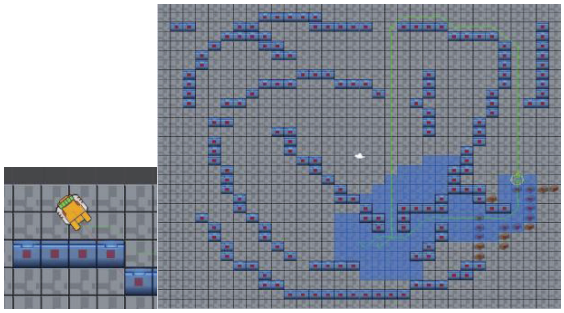


Fig. 4. Simulation environment example

B. Running Simulations

For training and testing our SESS-GAN model, we needed a high-fidelity simulation environment that could accurately represent the physics of the generated environments. After considering several options, we decided to use Unity due to its powerful physics engine and high level of fidelity. Unity is a popular game engine that supports high-quality graphics, physics simulations, and advanced AI. We used the headless mode of Unity to run our simulations without any graphical user interface, which significantly improved the speed of the simulation. Furthermore, we were able to speed up the simulation time to achieve faster training of our model. The physics engine in Unity was particularly useful for our training as it allowed us to accurately model the physical interactions in the environment and simulate collisions between objects. By running simulations in Unity, we were able to obtain the fitness score of GAN models in the fine-tuning stage of our SESS-GAN model, which helped improve its performance and generalization capabilities. The use of Unity for our simulation environment provided us with a high-fidelity, accurate, and efficient platform to train and test our SESS-GAN model. Its physics engine allowed us to accurately model the physical interactions between objects in the environment, which was crucial for our training. The headless mode and simulation speed-up capabilities of Unity also helped us achieve faster training times.

The model and its initial training data are made available online[2] for further research and evaluation by the community.

IV. Evaluation

A. Evaluation Metrics

We define two types of metrics in our evaluation study: i) simulation metrics and ii) environment metrics. Simulation metrics are used to evaluate the quality of the generated environments in terms of safety and risk. They include *collision count*, which measures the number of collisions that occur between the robot and objects in the environment, and *proximity time*, which measures the amount of time the robot spends in close proximity to objects. The risk score is calculated as the product of collision count and the ratio of proximity time to simulation time. On the other hand, environment metrics are used to evaluate the diversity and novelty of the generated environments. These metrics include *tile KL divergence* [7], which measures the similarity between the distributions of tiles in the generated environments and the original environment, and *layout uniqueness* [7], which measures the uniqueness of the layout of the generated environments compared to the original environment. These metrics are important in ensuring that the generated environments are not only safe but also diverse and interesting for the robot to explore.

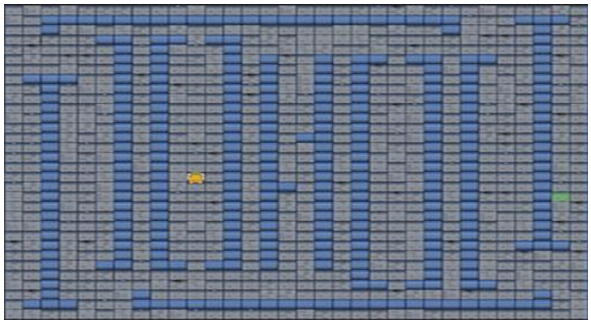


Fig. 5. Factory training environment

B. Input Training Environments

We consider three distinct testing scenarios to evaluate our solution. These include: i) a factory scene with autonomous robots, ii) a race track with a self-driving car, and iii) an area with regular and low obstacles where an autonomous drone is navigating. We aim to evaluate SESS-GAN's ability to learn and generate diverse and realistic simulation environments for these dynamic and complex scenarios. Through this evaluation, we aim to demonstrate the potential of SESS-GAN in generating high-quality environments and its fault-revealing power.

Factory Scene with Autonomous Robots: This environment involves multiple autonomous robots navigating through a factory setting (cf. Fig. 5). This environment is represented as a 32×32 grid, including two discrete tokens used to encode terrain: i) Ground (G): traversable space for robots, and ii) Wall (W): obstacles that block movement. This can be challenging for the robot control software to navigate through various obstacles and avoid collisions with other robots. Additionally, the robot agent must be able to complete tasks assigned to it within the factory, such as picking up objects, moving them, and placing them in the correct location.

[2] https://github.com/Charbel199/SESS-GAN

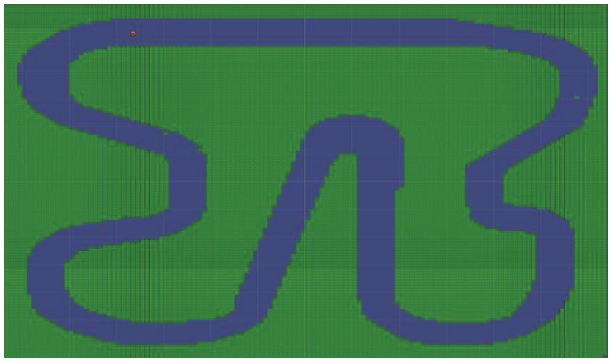


Fig. 6. Racetrack training environment

Race Track with Self-Driving Car: The second testing scenario involves a self-driving car navigating through a race-track (cf. Fig. 6). The race track environment is modeled as a 128×128 grid, providing higher resolution for fine-grained path planning. It uses two tokens: i) Track (T): the valid path the car can drive on, and ii) Out of Bounds (O): non-drivable zones where the car must not enter. The robot control software must navigate through the track at high speeds, while avoiding obstacles and other cars.

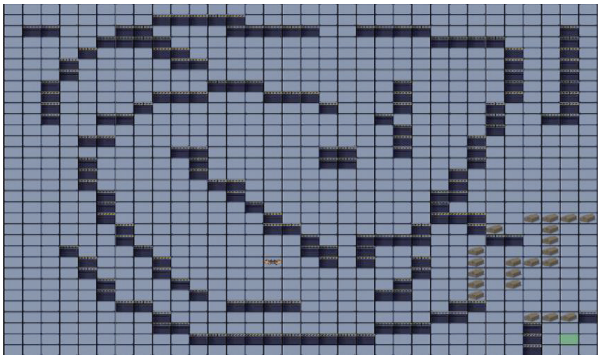


Fig. 7. Drone training environment

Autonomous Drone with Obstacles: The third testing scenario involves an autonomous drone navigating through an area with both regular and low obstacles (cf. Fig. 7). The drone testing area is encoded as a 32×32 grid, using three tokens: i) Ground (G): open areas that can be flown over, ii) Wall (W): tall obstacles that must be fully avoided, and iii) Ground Obstacles (O): low-height objects that the drone can fly over, but must still be detected and considered during path planning. This will require the robot control software to navigate through tight spaces and make quick decisions to avoid obstacles. The ability to fly over low obstacles provides an interesting challenge for the robot agent to navigate through the environment efficiently.

Overall, these three testing scenarios provide a diverse range of challenges for the SESS-GAN generator, from navigating through tight spaces, avoiding obstacles, to completing tasks assigned within a factory setting. We evaluate the output environments based on the metrics introduced previously.

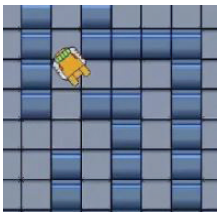
C. Types of Discovered Faults

In the course of our simulations, a fault in the obstacle avoidance system was detected and is illustrated in Fig. 8.1, wherein a flying agent collides with a dummy flying agent as it was not within its field of view. As depicted in Fig. 2, the obstacle avoidance algorithm utilized is a basic projection along the velocity direction, which may result in the neglect of any moving obstacles that approach from a wider angle, such as from behind or at an obtuse angle.

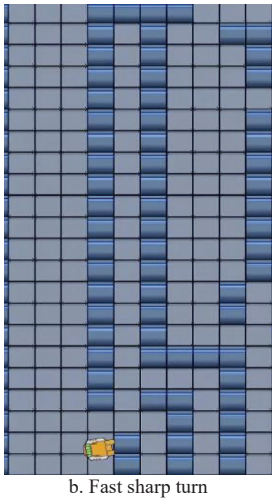
In the second identified fault, illustrated in Fig. 8.b, the agent under observation had previously been travelling from the upper part of the map, and had subsequently accumulated significant velocity. Upon encountering a sharp turn, the agent failed to decelerate in time, resulting in a collision with the surrounding wall. This demonstrates a problem with the agent's braking system.

In Fig. 8.c, we present the third fault discovered in our simulation environment. The agent in question is required to navigate through a narrow space with multiple turns. Due to its actuation mechanism, the agent is unable to turn in place and must move to execute a turn. As a result, the agent collides several times with the surrounding walls while trying to navigate through the narrow path.

a. Narrow Field of View



c. Narrow space turns



b. Fast sharp turn

Fig. 8: Types of detected faults

D. Output Environment Results

After training SESS-GAN in three different testing scenarios, we generated output environments to evaluate the model's performance. These output environments were generated by feeding the trained SESS-GAN model with noise vectors as inputs, which the model then transformed into realistic-looking synthetic environments that mimic the characteristics of the original training environments. By generating these output environments, we aim to analyze the model's ability to capture and replicate the visual and structural features of the original training environments, and to assess the model's generalization and robustness in generating diverse and realistic-looking synthetic environments. In the following part, we will discuss in detail the output environments generated by SESS-GAN for each of the three training environments, and we will provide visual and quantitative analysis of the model's performance.

Fig. 9 shows output environments generated from training the SESS-GANs on the three different training scenarios. For each training environment, we will show four output environments representing the output of genetic evolution tuning after: 0, 50, 100, and 200 generations.

Factory Scene with Autonomous Robots: The factory environment consists of autonomous robots navigating through a factory scene. The goal is to generate an environment that challenges the robots to navigate through it, revealing any faults or weaknesses in their navigation algorithms. As the SESS-GAN is trained on this environment, the generated environments become more complex, with smaller gaps and tighter spaces, making it increasingly difficult for the robots to navigate through. In the initial output environment after 0 generations, the gaps are relatively wide and the environment is relatively simple. However, after 50 generations, the gaps become smaller, requiring more precise navigation. By 100 generations, the environment becomes even more complex, with narrow passages and tight turns. After 200 generations, the environment is highly challenging, with very small gaps that require precise navigation. This progression of environments reveals the faults and weaknesses in the robot's navigation algorithms, allowing improvements to be made.

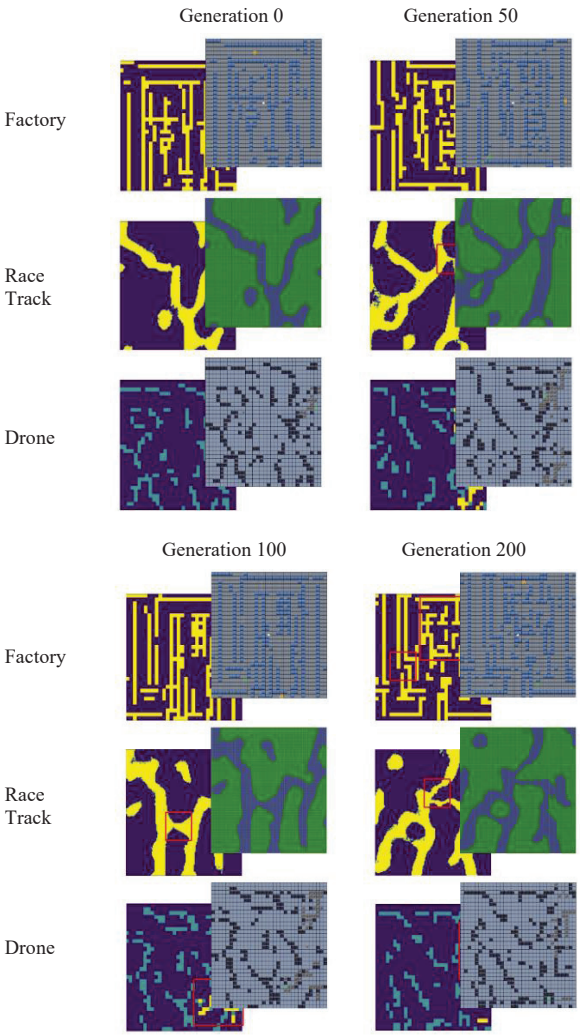


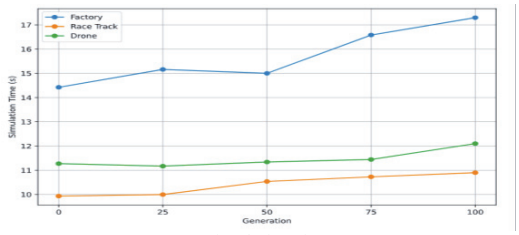
Fig. 9: Output environments

Race Track with Self-Driving Car: The racetrack environment consists of a self-driving car navigating through a racetrack. The goal is to generate an environment that challenges the car's navigation algorithms, revealing faults or weaknesses. As the SESS-GAN is trained on this environment, the generated environments become more challenging, with sharper corners and smaller passages. In the initial output environment after 0 generations, the environment is relatively simple, with wide turns and straightaways. However, after 50 generations, the corners become sharper and the passages become narrower. By 100 generations, the environment becomes even more challenging, with tight turns and very narrow passages. After 200 generations, the environment is highly complex, with very sharp turns and extremely narrow passages. This progression of environments reveals the faults and weaknesses in the car's navigation algorithms, allowing for improvements to be made.

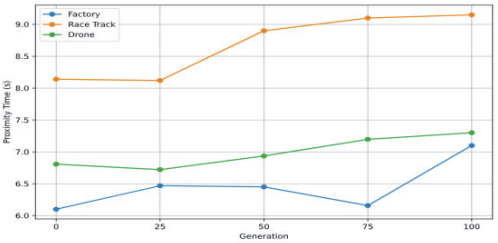
Autonomous Drone with Obstacles: The drone environment consists of an autonomous drone navigating through an area with regular and low obstacles that it can fly over. The goal is to generate an environment that challenges the drone's navigation algorithms, revealing faults or weaknesses. As the SESS-GAN is trained on this environment, the generated environments become more complex, with sections that can only be accessed by the drone. In the initial output environment after 0 generations, the environment is relatively simple, with few obstacles and open spaces. However, after 50 generations, the environment becomes more complex, with obstacles that the drone must fly over or around. By 100 generations, the environment becomes even more challenging, with sections that can only be accessed by the drone and more obstacles to navigate around. After 200 generations, the environment is highly complex, with many obstacles and areas that can only be accessed by the drone. This progression of environments reveals the faults and weaknesses in the drone's navigation algorithms, allowing for improvements to be made.

E. Simulation Metrics Evaluation

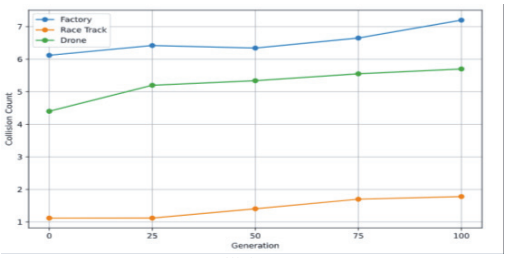
Results in Fig. 10 illustrate proximity time, collision count, and simulation time metrics across generations of our algorithm. Fig. 10.a depicting *simulation time* vs. generation, shows a general upward trend, indicating that as the algorithm progresses, it produces increasingly complex environments that require more time to simulate. This aligns with the objective of evolving environments that challenge the robot agents more intensively. Fig. 10.b tracks *proximity time*, measuring the cumulative time agents spend near obstacles without colliding. This metric helps quantify near-miss behavior, which reflects tighter navigation and higher environmental complexity. Fig. 10.c. shows the *collision count* per generation, offering insight into how often agents fail to avoid obstacles. A rising trend here may suggest increasingly difficult environments, while a decrease could indicate improved agent performance or conservative navigation strategies. Together, these graphs provide a comprehensive view of how environment difficulty evolves throughout the generations. For instance, the average number of collisions between generation #0 and generation #100 went from 3.9 to 7.5, the average proximity time went from 6.77 seconds to 7.87 seconds, and the average simulation time went from 11.9 seconds to 13.43 seconds, producing an average risk score between 2.21 to 4.39. Therefore, we can confirm that our initial fine-tuning led to GANs which generate 'harder' environments where the robot control softwares fail or near-fail more often



a. Simulation time



b. Proximity time



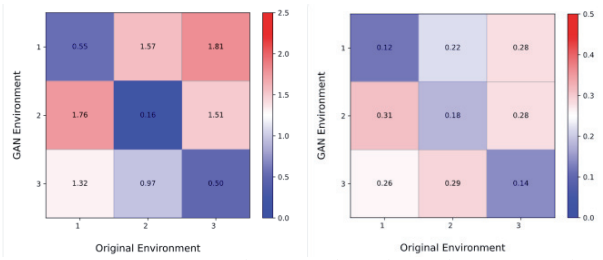
c. Collision count

Fig. 10. Simulation metrics evaluation results

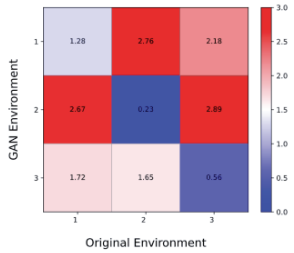
F. Environment Metrics Evaluation

Results in Fig. 11 show the TPKL divergence results in the form of 3×3 grids, where each cell represents the divergence between environments generated by a specific training set and a reference environment type. The diagonal elements reflect the divergence

between generated environments and the corresponding environment they were trained on. As expected, these diagonal values are consistently lower, indicating that the generative model successfully captures and reproduces the structural characteristics specific to each environment type. In contrast, the off-diagonal elements—representing divergences between generated environments and other original environments types, and they tend to show higher values, confirming that the generated environments are distinct. This matrix structure provides strong evidence that the generator is meaningfully learning type-specific distributions rather than converging to a single overfitted representation.



a. Factory use case matrix b. Racing track use case matrix



c. Autonomous drone use case matrix

Fig. 11. TPKL divergence results

In addition to TPKL divergence, and inspired by the evaluation approach in TOAD-GAN [2] , we assess structural variability by evaluating the uniqueness of the sampled patches. To do, we compute the uniqueness of square patches within our generated environments. Specifically, we sample 8×8 and 16×16 patches for each of the simulation environments and compute uniqueness percentages accordingly. Across all environments, the generated layouts show high average uniqueness scores. The factory and drone scenarios achieve over 91% average uniqueness, indicating a strong diversity in their generated environments. The racetrack scenario, while slightly lower at 81.14%, still maintains a high level of variation, especially at higher resolutions. This shows that we are able to produce distinct environment layouts, avoiding redundancy across generations

Table 1. Structural variability through uniqueness results

	Sample patches		Avg. Uniqueness
	8x8	16x16	
Factory	82.69 %	100 %	91.34 %
Race track	66.67 %	95.60 %	81.14 %
Drone	33 %	98.20 %	91.27 %

G. Discussion

To sum up, our experiments show that SESS-GAN effectively generates environments that increase in complexity across generations, as indicated by rising simulation time, proximity time, and collision count. TPKL divergence results confirm that the model captures environment-specific structures, and uniqueness scores highlight strong layout diversity. These outcomes support the model's ability to produce fault-revealing simulations for control software testing.

Due to the uniqueness of our study, a direct comparison with existing approaches was not feasible. To the best of our knowledge, this is the first study to conduct such simulation experiments in this specific context. As such, we identify the development of appropriate benchmarks and comparative methodologies as an important direction for future work.

V. Conclusion

This paper introduces the SESS-GAN model which leverages single-shot tokenized GANs to generate a preliminary model that is then fed into an evolutionary algorithm for fine-tuning. Our work promises significant implications for the development and testing of robotic control software, as the use of SESS-GAN can create simulations that are more effective at identifying potential flaws in control software, aiming to produce safer and more reliable autonomous systems. The model is made available online for further research and evaluation by the community.

We are currently extending SESS-GAN's evaluation by testing on larger and more complex environments, considering different types of control systems, e.g., [17, 18]. Subsequently, we plan to test the model on different kinds of control algorithm software, to assess the model's robustness and generalization capabilities, and potential for different types of control systems.

References

[1] Arnold J. and Alexander R., *Testing Autonomous Robot Control Software Using Procedural Content Generation.* International Conference on Computer Safety, Reliability, and Security (SAFECOMP'13), 2013. 33-44.

[2] Awiszus M., Schubert F., and Rosenhahn B., *TOAD-GAN: Coherent Style Level Generation from a Single Example.* AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2020. pp. 10-16.

[3] Beckham C. and Pal C. J., *A step towards procedural terrain generation with GANs.* CoRR abs/1707.03383, 2017.

[4] Bengesi S., et al., *Advancements in Generative AI: A Comprehensive Review of GANs, GPT, Autoencoders, Diffusion Model, and Transformers.* IEEE Access, 2024. 12: 69812-69837.

[5] da Silva B., Maia J., and de Carvalho W., *Procedural content generation in pervasive games: state of affairs, mistakes, and successes.* International Journal of Pervasive Computing and Communications, 2024. 20(3): 345-364.

[6] Dash A., Ye J., and Wang G., *A Review of Generative Adversarial Networks (GANs) and Its Applications in a Wide Variety of Disciplines: From Medical to Remote Sensing.* IEEE Access, 2024. 12: 18330-18357.

[7] Dutra P., Villela S., and Neto R., *A mixed-initiative design framework for procedural content generation using reinforcement learning.* Entertainment Computing, 2025. 52: 100759.

[8] Edwards M., Jiang M., and Togelius J., *Search-Based Exploration and Diagnosis of TOAD-GAN.* Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE'21), 2021. 140-147.

[9] Ghelfi E., et al., *Adversarial Pixel-Level Generation of Semantic Images.* CoRR abs/1906.12195, 2019.

[10] Humeniuk D., Khomh F., and Antoniol G., *A Search-Based Framework for Automatic Generation of Testing Environments for Cyber-Physical Systems.* Information & Software Technology, 2022. 149: 106936.

[11] Pan C. and Yao X., *Neural Architecture Search Based on Evolutionary Algorithms with Fitness Approximation.* IEEE Inter. Joint Conference on Neural Network (IJCNN'21), 2021. 1-8.

[12] Paria B., Lahiri A., and Biswas P., *PolicyGAN: Training generative adversarial networks using policy gradient.* International Conference on Advances in Pattern Recognition (ICAPR'17), 2017. 1-6.

[13] Sarkar A., et al., *Procedural Content Generation via Knowledge Transformation (PCG-KT).* IEEE Transactions on Games, 2024. 16(1): 36-50.

[14] Shaham T., Dekel T., and Michaeli T., *SinGAN: Learning a Generative Model From a Single Natural Image.* IEEE International Conference on Computer Vision (ICCV'19) 2019. 4569-4579.

[15] Torrado R., et al., *Bootstrapping Conditional GANs for Video Game Level Generation.* IEEE Conference on Games (CoG), 2020. 41-48.

[16] Wang S., et al., *Learning Regularity for Evolutionary Multiobjective Search: A Generative Model-Based Approach.* IEEE Computational Intelligence Magazine, 2023. 18(4): 29-42.

[17] Noueihed H. et al., Knowledge-based virtual outdoor weather event simulator using unity 3D. J. Supercomput., 2022, 78(8): 10620-10655

[18] Noueihed H. et al., Simulating Weather Events on a Real-world Map using Unity 3D. SMARTGREENS 2022: 86-93