Realistic 3D Drone Simulation with Path-Planning in Unreal Engine 5

Irina Hallinan irina_hallinan@berkeley.edu
Tianyun Yuan 7ianyun@berkeley.edu
Buyi Geng uyikatarina@berkeley.edu
Xinyu Deng xinyudeng@berkeley.edu
CS 184/284A, University of California, Berkeley

Figure 1: Drone model flying above the model of the Palace of Fine Arts in Unreal Engine 5

ABSTRACT

Simulating realistic drone flight could be helpful for drone operators as a training and testing ground. Existing drone simulators such as Microsoft AirSim don't have 3D world scenes of any location on Earth. Our goal is to proof-of-concept that provides a realistic scenario of flying a drone in a complex scene, approximating the real world. The novelty of our approach is in combining complex environment with physics-based realistic simulation and path-planning in a free game and physics engine, Unreal Engine 5. Specifically, we focus on 1 realistic scenario. We compared different path-planning algorithms to select the best flight path. Additionally, we increased the simulation realism by animating collisions or crashes. The final video shows the drone flying through the 3D world around obstacles through predefined path in space. The video can be seen on the website: https://irina694.github.io/3d-drone-sim/.

KEYWORDS

simulation, 3D, physics, Unreal Engine 5, path-planning

ACM Reference Format:
Irina Hallinan irina_hallinan@berkeley.edu, Tianyun Yuan 7ianyun@berkeley.edu, Buyi Geng uyikatarina@berkeley.edu, and Xinyu Deng xinyudeng@berkeley.edu. 2023. Realistic 3D Drone Simulation with Path-Planning in Unreal Engine 5. In *Proceedings of 3D Drone Simulation (CS 184/284A).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

CS 184/284A, May 2023, UC Berkeley

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Small Unmanned Aerial Vehicles (UAVs), or drones, have many applications ranging from search-and-rescue missions, to disaster responses to cinematography [1], [2]. The usage of drones rose in popularity due to recent advances in robotics, computer vision, and artificial intelligence [3]. Every year, drones are getting better at

navigating more complex environments autonomously, and in small fleets, by communicating with each other [4]. Drones are currently used for many tasks, such as taking aerial photographs, inspecting bridges, and documenting warehouse inventory. Simulating drones can offer an advantage to piloting a real drone. For example, drone pilots can prepare for a specific mission by learning the route. Simulations are also less expensive and has a reduced risk. In the United States, flying a drone in the real world needs to adhere to the rules of the Federal Aviation Administration [5]. It can also be hazardous due to environmental factors (i.e., wind, rain, birds, lightning conditions), and inability of algorithms to land the drone safely.

In order for simulations to be useful in the real world, the simulation needs be as realistic as possible. With this goal in mind, we built a 3D drone flight simulator in Unreal Engine 5 that incorporates open-source data from Google Earth. Our application is a proof-of-concept that any place on Earth could be modeled and incorporated into the drone simulation given the requisite imagery. We also compared several drone path-planning algorithms that could inform the pilot about possible fly routes.

2 BACKGROUND

Historically, plane and space shuttle pilots train on simulators, approximating the real experience of flying before flying a real plane or a space shuttle. Similarly, drone simulators are becoming more ubiquitous as the number of drones and drone pilots increases. With the advancement of computer graphics, the realism of simulations is improving as well. There are a number of existing drone and flight simulators that approximate realistic scenarios, such as drone swarm simulator in Matlab [6] or a drone flight for nature conservation [7]. There are many open-source and proprietary tools for different scenarios and types of UAVs [8].

2.1 Previous Works

One widely used simulation tool is the Microsoft AirSim [9]. The software is open-source and is built in Unreal Engine, like our simulator. This software was released in 2017 and has developed many specialized capabilities, such as changing the hardware configuration of the drone. The tool allows the user to experiment with various machine learning techniques, using computer vision and reinforcement learning [10]. In addition to simulating flight, the AirSim tool can simulate autonomous cars.

Existing drone simulators such as Microsoft AirSim don't come with scenes based on real-world locations out of the box. Rather than using existing tool like AirSim, we decided to built our own tool. Our goal was to provide a proof-of-concept scenario of flying a drone through a 3D model of a real scene. We decided to build our own tool because learning AirSim and extending it for our use case would have been more time-consuming than building a tool from scratch.

2.2 The Novelty of Our Approach

Our approach to the scene modeling is different than AirSim because we focus on the realism of both the drone and the scene, whereas AirSim focuses on various types of drones and reinforcement learning. Our simulation aims to be physically accurate by using drone rotors approximating how certain types of real drones are made and the physics of their flight. Potentially, our tool could be used to plan a flight path of a real drone in any location, given available data, such as Google Earth satellite images.

3 MODELING REALISTIC ENVIRONMENTS

In order to create a realistic 3D model of an existing place, we experimented with available tools including Google Maps and Google Earth Studio. We chose a scene that the drone could potentially be flown in and that had interesting topographical features, but that wasn't available in existing flight simulators. Using Google Maps to capture a set of 3D images produces a low-resolution model. We then switched to the Google Earth Studio platform. We chose the Palace of Fine Arts in San Francisco as our example scene and its surrounding area. Google Earth Studio allows the user to capture an aerial video footage by following a specified camera path. We learned that spiral path doesn't work because it made it difficult to reconstruct a 3D model from a changing perspective. The Orbit type project worked best for our use case.

First, we captured the video from above the palace by slowly rotating the virtual camera, while keeping the top of the palace stationary. The total duration was 120 seconds. We then rendered the video as an MP4 movie file with 3601 frames at 30 fps. Rendered every 25th movie frame into a PNG image at 100% resolution using Blender. Second, we uploaded 145 PNG images into Meshroom open-source photoprogrammetry software to process and extract the 3D model and produce a dense 3D mesh of the scene. Overall, the Meshroom 3D model pipeline includes automatic feature extraction, which uses dspsift algorithm, image matching, feature matching, structure from motion, depth map calculation, meshing, and texturing. Finally, after we created a 3D mesh of the scene with 542,006 elements, we joined all mesh elements in Blender to have 1 object. We then exported the mesh file from Blender as glTF 2.0 into Unreal Engine. Figure 2 shows a screenshot from capturing data in Google Earth Studio.

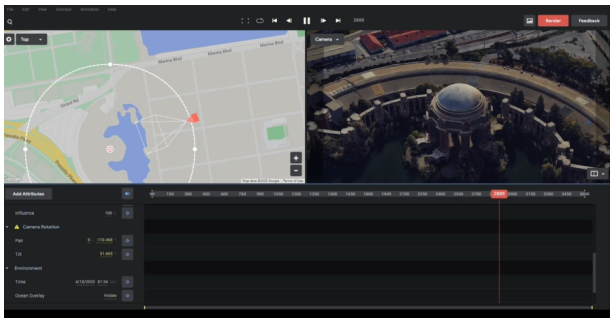


Figure 2: The Palace of Fine Arts in Google Earth Studio

4 IMPLEMENTATION IN UNREAL ENGINE 5

We chose Unreal Engine 5 (UE5) because it has a built-in capabilities to simulate realistic physics, light and shadow effects, and high-definition graphics. Unreal Engine 5 is a modern open-source game development engine. We integrated the scene model into the engine by creating a new level with objects representing the actors in our

simulation, including the scene and the drone. The programming language in Unreal Engine is C++. It also comes with a visual editor with powerful features, where users can drag and drop events and create custom code that interacts with in-game and level objects. We used a combination of object-oriented C++ classes and visual-style Blueprints. Figure 3 shows a Blueprint level-based event graph we created.

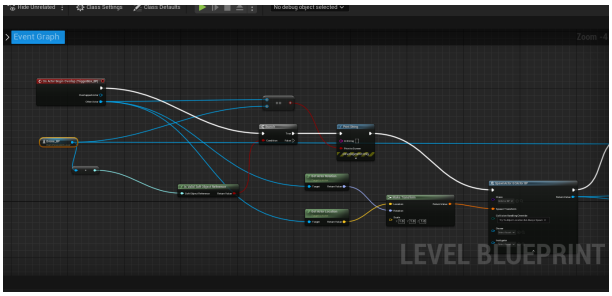


Figure 3: A part of the Blueprint graph we created in UE5

Additionally, UE5 offers comprehensive tutorials for beginners and includes a marketplace of assets that can be used by developers. The engine allows for easy API integration and has a less steep learning curve compared to other physical simulation frameworks. We discuss the details of our implementation in UE5 in the next section.

4.1 Drone Model

Using an open-source mesh, we modeled the drone's physics after a real-life quadcopter. The drone has 4 rotors attached to each of its corners, each providing varying thrusting force to the drone, allowing the drone to move, yaw, and pitch in any direction.

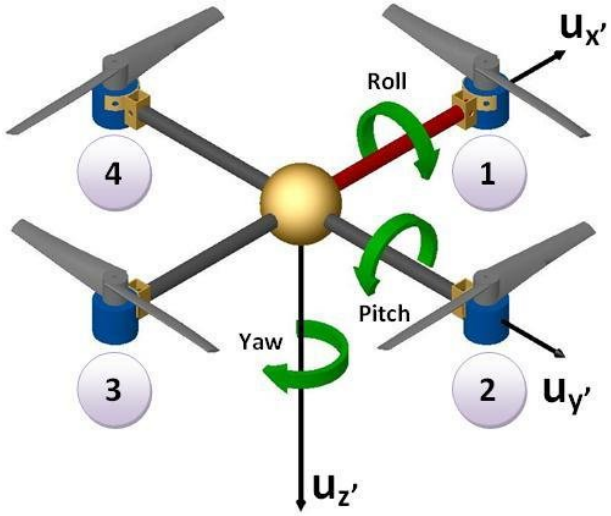


Figure 4: Quadcopter Design

4.2 Drone Controller

The path-planning algorithm passes the desired path as a list of 3d coordinates to the drone, the drone then goes from one coordinate to another. While giving delta displacement to the drone tick allows it to move smoothly, it does not provide an ideal simulation of a quadcopter, whose movement results solely from its rotors. Therefore, we implemented the P.I.D controller that provides a physically accurate controlling mechanism to the drone.

P.I.D. controller is a heuristic widely used in the robotics industry. Five PID controllers of the drone—three for x, y, and z coordinates, and two for yaw and pitch—create a feedback loop of errors and deltas, allowing the drone to smoothly adjust each one of the four rotors' thrusting power, approaching its target without human control.

Each PID controllers require 3 hyper-parameters: P, I, and D, small changes in one leading to huge differences in the controller's reaction to errors. Like real-life drone robotics, fine-tuning PID controllers involves repetitive trial and error. The drone being in a physical simulation, however, rendered the process much simpler.

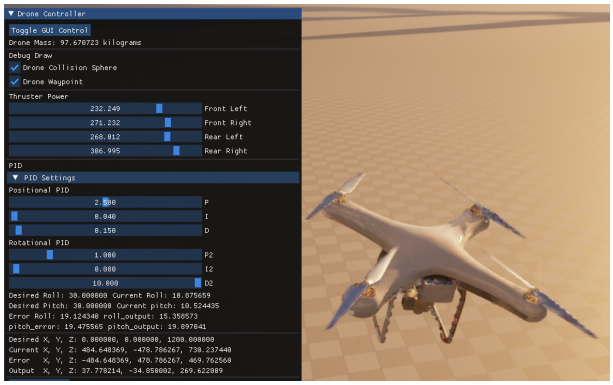


Figure 5: Fine-tuning PID through GUI

The outputs from the PID controllers are then weighted and passed to the calculation of thrusting power. The calculation goes as follows:

- **X pid** X pid is directly related to rear thrusting power, and inversely related to front thrusting power.
- **Y pid** Y pid is directly related to front thrusting power, and inversely related to rear thrusting power.
- **Z pid** Z pid is directly related to all rotors' thrusting power.
- **Pitch pid** Pitch pid is directly related to rear thrusting power, and inversely related to front thrusting power.
- **Yaw pid** Yaw pid is directly related to left thrusting power and inversely related to right thrusting power.

The rationale behind this intuitively makes sense: the drone likes to tilt forward when the waypoint is far in front, or go upward when the waypoint is high up. Yaw and pitch PIDs effectively balance the tilt angle of the drone, providing counterforce to prevent the drone from over-tilting.

4.3 Simulating Destruction

To add realism to the simulation, we implemented a failure mode when the drone crashes into an obstacle. Our implementation applies mass-based damage to the drone model, using UE5 Chaos Destruction system. Chaos Destruction system simulates physics-based object (static or skeletal) mesh destruction based on predefined force applied, which triggers a level of fracture that we can set: the more force is applied, the more the mesh fractures. This system allows us to achieve a crash effect upon collision by spawning a pre-fractured static mesh in place of the drone mesh when a collision event is detected. We also simulate the force applied to the drone by the wall/floor that it collides with.

4.4 Camera Perspectives

We used several perspectives to achieve various simulation effects. One camera follows the drone, implemented as a spring arm. This camera shows a 3rd person view of the scene. Another camera sits on top of the drone and shows a 1st person view of the scene, which offers an immersive view. Additionally, to produce cinematic collisions, we added a 3rd person character camera, which can be moved by the user to view the drone from different view points and perspectives, akin to 1st person games where the camera follows where the player is looking.

Figure 6: Immersive First-Person View

4.5 Obstacle Detection

We used a simple approach to real-time obstacle detection and avoidance approach. The drone's current location is tracked using its pawn's location in the game world. The drone controller then starts searching for obstacles in the vicinity of the drone by checking for floating boxes that were previously spawned in the game world. Once an obstacle is detected, the controller calculates the distance between the obstacle and the drone and determines if the obstacle is within a predefined range. If the obstacle is within range, the controller uses the drone's start and end location to calculate a path that avoids the obstacle. The new path is then inserted into the drone's current navigation plan, allowing the drone to adjust its course to avoid the obstacle. This approach helps the drone avoid collisions with obstacles in its path and maintain its trajectory towards the destination.

5 PATH PLANNING

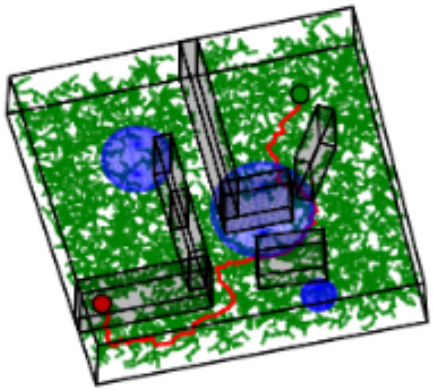
Path planning is a critical component in autonomous robotic systems, including drones, that involves determining a feasible and optimal route from a starting point to a destination while avoiding obstacles. It plays a significant role in drone navigation as it ensures safe and efficient operation in complex environments. By employing effective path planning algorithms, drones can navigate through cluttered spaces, minimize energy consumption, and accomplish tasks like aerial photography, package delivery, or search and rescue operations.

5.1 Interfaces with Map Reading& Drone Control

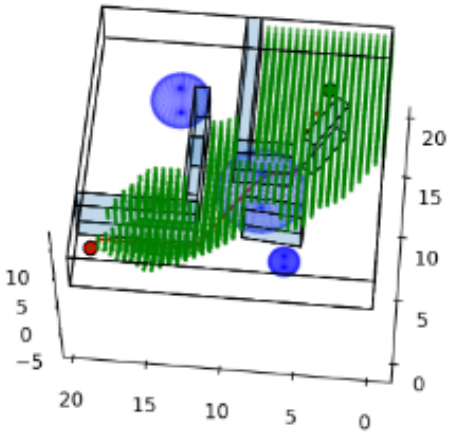
- **Map Reading:**
  After getting the map3D.py file, we get a JSON file for the algorithm input.
- **Drone Control:** After conducting the path planning Algorithms, we get a list of vectors to control the drone.

5.2 Path Planning Algorithm conducted

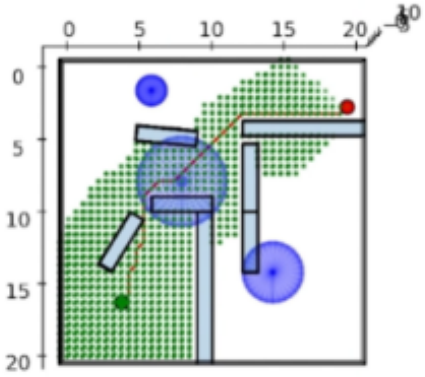
- **RRT-3D (Rapidly-Exploring Random Trees in 3D):**
  Probabilistic: The algorithm randomly samples the configuration space to create a tree that explores the environment. Incremental: The tree is constructed incrementally, one vertex at a time, which allows the algorithm to operate in real-time for certain applications. Informed exploration: RRT-3D balances exploration and exploitation by biasing the search towards unexplored areas, enabling efficient search in high-dimensional spaces.
- **RRTConnect-3D (Rapidly-Exploring Random Trees Connect in 3D):**
  Bidirectional search: The algorithm grows two trees, one from the start and one from the goal, and attempts to connect them in the middle. Goal-biased sampling: RRTConnect-3D uses a goal-biased sampling strategy to improve the likelihood of a successful connection between the trees. Faster convergence: Compared to RRT-3D, RRTConnect-3D often converges faster to a solution because of its bidirectional search approach.
- **A* (A-star) algorithm:**
  Heuristic-based search: A* uses a heuristic function to estimate the cost from the current node to the goal, guiding the search towards the goal more efficiently. Optimal solution: A* guarantees an optimal solution if the heuristic function is admissible (never overestimates the true cost) and consistent (satisfies the triangle inequality). Complete search: A* is a complete search algorithm, meaning that if there is a solution, it will find it.
- **Bidirectional A* algorithm:**
  Simultaneous search: Bidirectional A* simultaneously searches from the start and goal nodes, expanding nodes in both directions. Faster than unidirectional A*: This algorithm typically finds a solution faster than the unidirectional A* algorithm because it explores less of the search space. Optimal solution: Like A*, Bidirectional A* also guarantees an optimal solution if the heuristic function is admissible and consistent.



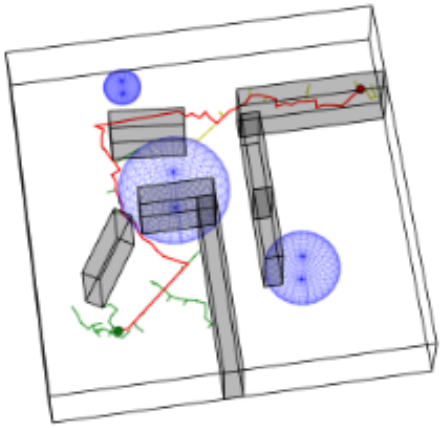
[RRT_3D]



[biodirectional AStar]



[AStar]



[RRTConnect3D]

5.3 Path Smoothing

Path smoothing is a post-processing technique applied to the output of a path planning algorithm to reduce the number of waypoints and improve the overall quality of the path. One way to achieve path smoothing is by adjusting the step size during the path generation process.

Here is our approach to path smoothing using variable step size:

- Start with an initial path: Generate a path using your preferred path planning algorithm. This path might contain many waypoints and sharp turns.
- Define a maximum step size: Determine the maximum distance allowed between consecutive waypoints in the smoothed path. This step size should be large enough to reduce the number of waypoints, but small enough to maintain the required accuracy and avoid collisions with obstacles.
- Initialize the smoothed path: Create an empty list to store the waypoints of the smoothed path. Add the first waypoint of the initial path to the smoothed path.
- Iterate through the initial path: For each consecutive pair of waypoints (A and B) in the initial path, perform the following steps:

  a. Calculate the distance between the current waypoint (A) and the next waypoint (B). If the distance is less than or equal to the maximum step size, add waypoint B to the smoothed path.

  b. If the distance between waypoints A and B is greater than the maximum step size, calculate the number of intermediate waypoints required to maintain the desired step size between waypoints. For instance, if the distance between A and B is 1.5 times the maximum step size, you'll need to insert one additional waypoint between A and B.

  c. Generate intermediate waypoints by interpolating between A and B, using linear interpolation or a more advanced method like cubic splines, depending on the desired smoothness. Add the intermediate waypoints to the smoothed path.
- Add the final waypoint: Once reached the end of the initial path, add the last waypoint to the smoothed path.

We successfully achieved the goal of path-smoothing by adjusting the step size smaller around 2.7(previously 0.5), which reaches the best balance for drone control as well as a smoother path due to the negative effect small step size has on drone control.

5.4 Comparison between algorithms

We selected the RRTConnect-3D algorithm for path planning due to several advantages. First, compared to RRT-3D, RRTConnect-3D exhibits faster convergence, making it possible to identify viable paths more quickly. Second, because its trees grow more purposefully toward one another, RRTConnect-3D typically produces smoother paths than RRT-3D, leading to better path quality. Third, RRTConnect-3D can adapt to complicated environments and can handle a variety of project requirements and circumstances since it is scalable and well-suited for high-dimensional spaces. Finally, RRTConnect-3D outperforms A* and bidirectional A* in complex scenarios in terms of computation time, striking a balance between computational efficiency and path quality. Overall, using RRTConnect-3D gave us a path-planning solution for our drone in challenging conditions that was successful and efficient.

6 LIMITATIONS

The simulation as a proof-of-concept is complete. However, there are technical limitations to this work. The biggest challenges we encountered were the integration between Unreal Engine and Python used for path planning. Although UE5 offers an experimental Python API, the feature is not production ready and documentation was sparse. Additionally, since UE5 has many built-in capabilities, learning how to use its functionalities was challenging due to the amount of available options. Finally, the way we encode a complex scene into our program can be improved.

7 FUTURE WORKS

In the future, we would like to improve the integration of the components such as comparing different path planning algorithms visually in Unreal Engine. We could also improve the resolution of the scene model by getting higher resolution images. One way to accomplish a higher resolution scene mesh would be to take videos from different angles (closer and further away), to supplement the original video recording. Having more views of the same scene could make the model into a denser 3D mesh. Finally, improving the way collisions are detected is a possible future work that we didn't have time to explore in this prototype.

8 CONCLUSION

Creating drone simulation for various fields is important and we leveraged Unreal Engine 5 for implementing computer graphics concepts in a realistic simulation. Search-based algorithms for path planning in 3D space have potential applications in drone navigation and robotics. We showed comparison between multiple search-based algorithms in a 3D environment. A web version of this report with videos can be found on: https://irina694.github.io/3d-drone-sim/.

ONLINE RESOURCES

Blender v3.5, Google Earth Studio v1.7, Meshroom v2023.1.0, Unreal Engine v5.1.1 with UnrealImGui plugin v1.22 and Python API, Visual Studio 2022.

Search-based 3D algorithms GitHub library: https://github.com/zhmreal/PathPlanning/tree/master/Search$_{b}ased_{P}lanning/Search_{3}D$.

REFERENCES

[1] A. Mairaj, A. I. Baba, and A. Y. Javaid, "Application specific drone simulators: Recent advances and challenges," *Simulation Modelling Practice and Theory*, vol. 94, pp. 100–117, 2019.

[2] C. Huang, F. Gao, J. Pan, *et al.*, "Act: An autonomous drone cinematography system for action scenes," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7039–7046. DOI: 10.1109/ICRA.2018.8460703.

[3] A. Al-Kaff, D. Martin, F. Garcia, A. de la Escalera, and J. M. Armingol, "Survey of computer vision algorithms and applications for unmanned aerial vehicles," *Expert Systems with Applications*, vol. 92, pp. 447–463, 2018.

[4] D. Câmara, "Cavalry to the rescue: Drones fleet to help rescuers operations over disasters scenarios," in *2014 IEEE Conference on Antenna Measurements Applications (CAMA)*, 2014, pp. 1–4. DOI: 10.1109/CAMA.2014.7003421.

[5] K. W. Smith, "Drone technology: Benefits, risks, and legal considerations," *Seattle J. Envtl. L.*, vol. 5, p. 291, 2015.

[6] E. Soria, F. Schiano, and D. Floreano, "Swarmlab: A matlab drone swarm simulator," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 8005–8011. DOI: 10.1109/IROS45743.2020.9340854.

[7] E. Bondi, D. Dey, A. Kapoor, *et al.*, "Airsim-w: A simulation environment for wildlife conservation with uavs," in *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, 2018, pp. 1–12.

[8] A. I. Hentati, L. Krichen, M. Fourati, and L. C. Fourati, "Simulation tools, environments and frameworks for uav systems performance analysis," in *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*, 2018, pp. 1495–1500. DOI: 10.1109/IWCMC.2018.8450505.

[9] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics: Results of the 11th International Conference*, Springer, 2018, pp. 621–635.

[10] X. Liang, Y. Liu, T. Chen, M. Liu, and Q. Yang, "Federated transfer reinforcement learning for autonomous driving," in *Federated and Transfer Learning*, Springer, 2022, pp. 357–371.