Drone Navigation Using Deep Reinforcement Learning

Nishant Pandey
Master of Engineering in Robotics
University of Maryland
College Park, Maryland
npandey2@umd.edu

Jayasuriya Suresh
Master of Engineering in Robotics
University of Maryland
College Park, Maryland
jsuriya@umd.edu

***Abstract*—This paper details a drone obstacle avoidance system developed using Microsoft AirSim and the Deep Deterministic Policy Gradient (DDPG) algorithm. It aims to train drones autonomously for real-time navigation and obstacle avoidance. Leveraging realistic drone dynamics and sensor data from AirSim, and the DDPG algorithm's continuous control capabilities, the system trains drones using LIDAR or LIDAR along with a depth sensor. A robust training protocol within AirSim includes scripts for managing training and evaluation, demonstrating the drone's learning and performance in various scenarios. This project highlights the practical application of reinforcement learning in enhancing autonomous drone navigation, using complex sensor data integration and real-time response in simulation.**

***Index Terms*—DDPG, AirSim, Drone, LIDAR, Depth**

I. Introduction

When it comes to unmanned aerial vehicles (UAVs), obstacle avoidance poses a significant barrier to autonomous navigation in complicated situations. Conventional navigation systems have a strong reliance on pre-programmed routes and may not be able to interpret data in real-time, which is important for making dynamic decisions in congested or unfamiliar environments. Reinforcement learning (RL) has emerged as a possible solution to these problems, giving UAVs a framework to learn from interactions with their surroundings and make wise navigational decisions.

This work presents a system that combines the Deep Deterministic Policy Gradient (DDPG) algorithm with Microsoft's AirSim simulator to improve drone navigation ability. For the intricate requirements of UAV training, AirSim offers a realistic simulation environment with high-fidelity visuals and physics. Based on sensor data, including depth, perception, and LiDAR, the drone tries to learn the best navigation and obstacle avoidance techniques thanks to the DDPG algorithm, which is well-known for its effectiveness in high-dimensional, continuous action domains.

The drone learns to make judgments based on a constant stream of sensor data, including inputs from LiDAR and depth sensors, by utilizing reinforcement learning, leaning heavily on the DDPG algorithm. By using this technique, the drone's autonomous navigation capabilities are enhanced not only by being able to identify obstacles but also by figuring out effective maneuvers to avoid them and the best path to reach the goal.

This project has various industrial applications such as logistics, agriculture, and infrastructure inspection, where drones need to navigate autonomously through cluttered spaces. For instance, in logistics, drones equipped with these capabilities can efficiently maneuver through warehouses or outdoor areas to transport goods. Overall, the enhanced obstacle recognition and avoidance capabilities provided by this project can lead to safer, more reliable, and more efficient autonomous drone operations across various industrial applications.

This project compares the applicability of different sensors (LiDAR and depth) and their combination using various evaluation metrics. Several important criteria are used in this system's evaluation to gauge how well the taught drone avoids obstacles. One of the main metrics is the collision rate, which measures how frequently the drone collides with objects while undergoing navigation trials. The navigation success rate too is another important indicator that assesses how well the drone can navigate to its intended location without running into any obstacles. The final metric used is the average reward per episode to indicate how well the drone performs as the testing of the trained model goes forward.

The following sections of this paper describe the methodology of how DDPG and Airsim are integrated with an explanation of the reward function utilized. Then our contributions are explained. The study of the results that were obtained from the testing and training are described under results followed by the challenges that were encountered. We summarize our findings along with future work that can be done to build upon our project.

II. Literature Survey

Significant progress has been made in the application of reinforcement learning (RL) to autonomous drones, especially with the use of algorithms like the Deep Deterministic Policy Gradient (DDPG) in Microsoft AirSim [7] and other simulation environments. This project offers a thorough basis for comprehending the intricacies and technological applications necessary for drone navigation systems that employ reinforcement learning to be successful.

The H. Tan [4] rigorously analyzes the DDPG algorithm, offering a theoretical and practical examination of its application

in RL environments. This paper highlights the effectiveness of DDPG in achieving sophisticated control strategies in high-dimensional action spaces, which is essential for the continuous and complex decision-making processes required in drone navigation. This project uses this paper [4] to understand the basics of DDPG and its working.

This project uses DDPG to create a model that is used for navigation to a designated goal. For this project, we wanted to use continuous action space as it makes the entire navigation process very smooth. However, multiple algorithms were explored before coming to this decision. The TD3 [8] algorithm was one of them. This algorithm addresses the approximation errors in the actor-critic network and is overall much better for quality output, but we stuck to DDPG due to simplicity, and higher computational Efficiency. Hyperparameter tuning is much simpler for DDPG as it has fewer parameters than TD3. Training times are lower as well. These factors played a huge role as the AirSim [7] itself is very resource-intensive.

To understand the whole implementation of projects with similar objectives we went through multiple papers notably the paper by Tu, Guan-Ting, and Juang, Jih-Gau [10]. This paper uses a well-known reinforcement learning state–action–reward–state–action (SARSA) algorithm for path planning and obstacle avoidance purposes for a drone. Similarly, the paper by Sang-Yun Shin, Yong-Won Kang, and Yong-Guk Kim [9] compared discrete action space RL algorithms and continuous action space algorithms for the application drone navigation and obstacle avoidance aided us a lot. Their study showed that continuous action space algorithms outperformed the discrete action space algorithm for this specific application, thus backing the decision of this paper to stick to a continuous action space algorithm.

The specific implementation of DDPG for drone obstacle avoidance in AirSim by John Venti [3] directly aligns with the focus of our project. This repository uses the same environment as the one used in the project. The algorithm used is DDPG along with ConvLSTM to navigate through the obstacles in AirSim. However, this repository only avoids obstacles in the height direction unlike our project which fixes the Z height and manipulates X and Y coordinates only.

Finally, in our research, we leveraged the GitHub repository "UAV Navigation with Deep Reinforcement Learning in AirSim" [6] by H. L. S. Dias for significant portions of our implementation, particularly focusing on the reward function, action space, and observation space design. The repository served as a valuable reference, offering code snippets and architectural insights that facilitated the development of our DDPG-based UAV navigation system. This repository unlike us used different RL algorithms like TD3, PPO, etc. They used a depth sensor for obstacle avoidance, whereas we included LIDAR data in the observation space as well.

III. Methodology

A. Environment Setup

Figure 1 shows that AirSim is configured with a quadrotor drone model and a simulated urban environment (Airsim Neighbourhood environment). The drone is equipped with depth sensors and LIDAR to perceive its surroundings. A single LIDAR sensor is placed on the bottom of the drone, AirSim provides the ability to change certain LIDAR parameters like LIDAR range, rotation per second, number of channels, and range of vertical and horizontal field of view. These parameters are varied to find the best fit for the project, considering the accuracy of obstacle detection (thin poles). The parameters used for the lidar are as follows:

Fig. 1. Quadrotor drone model spawned in AirSim neighborhood environment.

- Number of Channels are 16.
- Horizontal FOV is -90 to 90 degrees
- Vertical FOV is 10 degrees to -30 degrees.
- 30 Rotations Per Second

These were chosen after experimentation and hyper-tuning the values.

In another set of training experiments both LIDAR and depth sensors are used together to get a wider FOV and better obstacle avoidance. This combination combines the swiftness of Lidar-based observations while mitigating the inability of the Lidar to detect thin obstacles. The depth camera has a FOV of 90 degrees. The drone uses the NED(North East Down) system which is the common coordinate system for UAVs.AirSim also makes use of the same coordinate system. This allows for easy control of the drone using the AirSim Python API.

B. DDPG Implementation

For DDPG implementation, Stable Baselines3 (SB3) [5] which is a set of reliable implementations of reinforcement learning algorithms in PyTorch was used. The actor-network outputs continuous actions given a state input. It typically consists of fully connected layers and a tanh activation function to scale the output to the action space. Whereas, the critic-network inputs a state and an action and outputs the corresponding Q-value. It also consists of fully connected layers.

Stable baselines3 provides the option to either select one of many predefined neural networks(MLP, CNN, RNN, etc) or create a custom model. For this project, the MLP model is selected for learning optimal policies and handling continuous action spaces.

DDPG is an off-policy algorithm, which means it can learn from experiences generated by a policy other than the one now under optimization. This is made possible by keeping a replay buffer that saves previous experiences, allowing the model to sample and learn from them several times.

C. Action and observation space

To utilize the DDPG algorithm properly, it is important to use an accurate set of reward functions and a detailed understanding of action and observation space. In our project, action space defines the set of possible actions that the agent (drone) can take at each step.

For both sets of experiments with the LiDAR and LiDAR with Depth, the action space is continuous and includes two components: the velocity in the x-y plane (xy_{speed}) and the yaw rate (yaw_{speed}). The action space is defined as a Box space from the gym [11] library, which represents a continuous (real-valued) space where the velocity in the x-y plane has a range from -1 to 5 which can be adjusted.

The observation space includes features derived from the drone's state and LiDAR data for the LiDAR-only implementation. The state features include normalized distance to the goal and vertical distance to the goal, the LiDAR features include distances to obstacles in three directions (front, left, right). The observation space is also defined as a Box space, but the values are normalized between 0 and 1. The shape of the observation is a 1D array with a length equal to the sum of state feature length and LiDAR feature length.

The observation space for LiDAR and depth sensor implementation includes various features that describe the state of the drone, the environment, and information from the sensors (LiDAR and depth camera). The depth image features represent distances to objects in the environment, state features indicate the drone's relative position and orientation to the goal, and LiDAR features provide information about obstacles in three key directions.

D. Reward Function

The reward function is designed to guide the drone toward the goal while avoiding obstacles. The reward function consists of several components:

- **Goal Distance Reward:** The term `reward_distance` is computed based on the reduction in distance to the goal from the previous step to the current step. This encourages the drone to move closer to the goal. The coefficient `distance_reward_coef` scales this reward.
- **Position Punishment:** The term `punishment_pose` penalizes deviations from the direct path to the goal. It is computed as the normalized distance from the current position to the goal.
- **Obstacle Avoidance Punishment:** The term `punishment_obs` penalizes the drone for getting too close to obstacles.
- **Yaw Error Punishment:** The term `yaw_error_cost` penalizes large deviations in the yaw angle from the desired direction. This helps the drone maintain a stable orientation.
- **End Conditions:**
  - **Reward for Reaching the Goal:** A fixed reward `reward_reach` is given if the drone reaches the goal.
  - **Penalty for Crashing:** A fixed penalty `reward_crash` is given if the drone crashes.

The overall reward is calculated as:

$$\text{reward} = \begin{cases} \text{reward\_reach if the goal is reached} \\ \text{reward\_crash if the drone crashes} \\ \text{reward\_distance} - 0.1 \times \text{punishment\_pose} \\ -0.2 \times \text{punishment\_obs} - 0.1 \times \text{punishment\_action} \\ -0.5 \times \text{yaw\_error\_cost otherwise} \end{cases} \tag{1}$$

This reward function was taken from [6] and then modified by removing the z components in the equations. This was done to fix the Z height of the drone. So that the drone does not come up with a path that simply maneuvers around the obstacles by flying high above them as this is not the objective of the project. The drone should not escape the environment as well by flying very high.

E. Implementation

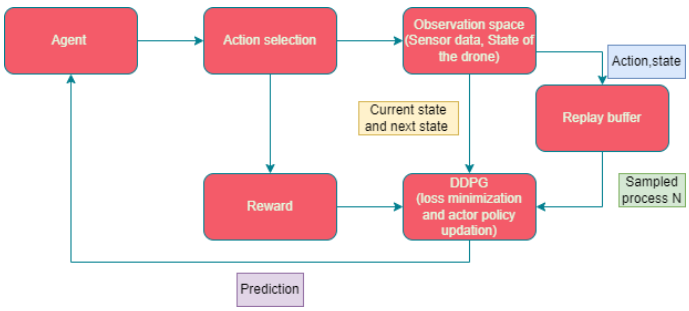

Fig. 2. Working Flow Diagram

Figure 2 shows the working of our implementation. The agent is the drone that performs actions in the XY plane (Z is fixed). The observation space consists of the sensor data and drone state information. The action generates a reward which along with all the other data is provided to the DDPG algorithm, where its actor-critic network provides its predictions back to the agent. The project involves training the drone with two different sensor setups: one utilizing both depth and LiDAR sensors, and the other relying solely on LiDAR sensor. For each setup, a goal location is defined in such a way that the drone's path includes a sufficient number of obstacles, creating a challenging environment for training. The training process is carried out for approximately 100,000 time steps. The same set of hyperparameters is used for both sensor setups, and these parameters have been fine-tuned to optimize performance. The selected hyperparameters are a batch size of 1280, a training frequency of 500, gradient steps set to 500, a buffer size of 50,000, a learning rate of 0.001,

learning starts at 2000 timesteps, and a gamma (discount factor) of 0.99. Noise is also included for the agent to explore the environment.

After completing the training, the models for each sensor setup are evaluated over 300-500 episodes. The performance of the models is assessed using several metrics: success rate (the percentage of episodes where the drone successfully reaches the goal without collisions), collision rate (the percentage of episodes where the drone collides with obstacles), and the average reward per episode (reflecting the efficiency of obstacle avoidance and goal achievement).

An important aspect to notice is that although the goal location of the evaluation remains the same, the criterion of success slightly changes. It is expected that the distance from the goal point of the drone will be about 5m without collision to declare the episode to be successful. This is done because reaching the exact point is not possible because of the nature of implementation itself and as we have carefully set the goal points very near to large obstacles. This was done keeping in mind that the drone should reach the goal and should not collide with obstacles regardless of goal location.

IV. Contributions

The project leverages multiple repositories, incorporating elements from each to create a distinct output. The repository from H. L. S. Das [6] employs a TD3 algorithm along with a depth sensor for navigation and goal-reaching. This repository served as our base, but we made numerous modifications to achieve our desired results. While both repositories utilize the stable-baseline3 [5] implementation of DRL algorithms, our implementation diverged by employing a DDPG algorithm instead of TD3. This alteration fundamentally changes the dynamics of our implementation. Although TD3 offers several improvements over DDPG, resulting in qualitative enhancements in their results, our approach benefits from using LiDAR or a combination of LiDAR with a depth sensor. This choice simplifies the process and reduces the time required for both training and evaluation.

Our project also drew inspiration from John-Venti's "DDPG-AirSim-Drone-Obstacle-Avoidance" [3], which uses DDPG for obstacle avoidance in a similar environment. However, their implementation focuses solely on height control, whereas our implementation operates in the xy-plane with a fixed z coordinate for the drone. Unlike their approach, which employs a ConvLSTM network with depth images, our implementation uses an MLP neural network and incorporates LiDAR data. This results in higher accuracy and a more efficient training process. While their results are slightly more generalizable due to the random nature of their goals and basic obstacle avoidance, our approach offers a more robust solution for our specific navigation and goal-reaching tasks. To implement these changes, there was heavy refactoring of the code. Observation space needs to be modified to take into account LiDAR data as well along with depth sensor data. Tweaking the LiDAR sensor parameters needs to be done as well to get accurate results and unlike many repositories that lack thorough quantitative analysis, our work tries to tackle this problem by providing necessary metrics for a better understanding.

V. Results

The results accumulated throughout this project compare multiple aspects of two different methods of implementation. Starting from their quantitative aspects to their qualitative aspect. Certain inferences are drawn from the results explaining which approach is better.

A. Quantitative results

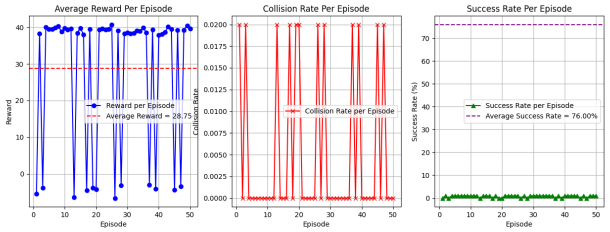

Fig. 3. Evaluation metrics of implementation using LiDAR sensor

The results presented in figure 3 include three metrics: average reward per episode, collision rate per episode, and success rate per episode. The graphs indicate that the LiDAR-only implementation achieves a relatively high overall success rate while maintaining a low collision rate (with only one collision per episode, as the episode ends upon collision). Additionally, the average reward remains quite consistent, although it occasionally drops below zero due to the substantial penalty for collisions.

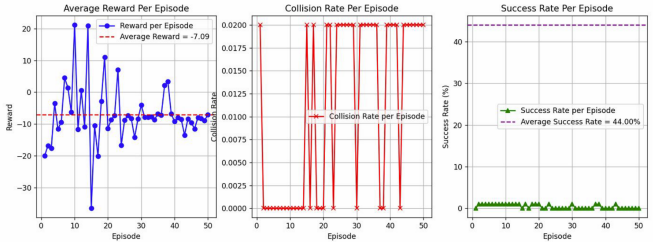

Fig. 4. Evaluation metrics of implementation using LiDAR and depth sensor

Similarly, the results presented in figure 4 include three metrics: average reward per episode, collision rate per episode, and success rate per episode. The graphs indicate that the LiDAR and depth sensor implementation achieves a relatively low overall success rate while maintaining a low collision rate (with only one collision per episode, as the episode ends upon collision). Additionally, the average reward is very inconsistent, the average reward is also very low, and the reward per episode is mostly close to zero. The graphs above show a testing-oriented comparison of both implementations. The results are very counter-intuitive, as it is assumed that more sensors may lead to better-generated models which is not happening here at all.

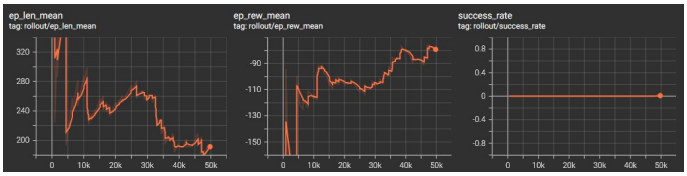

Fig. 5. Training metrics for multi-sensor setup

Training time for both cases is also vastly different. The training time for LiDAR-only implementation is much less than the LiDAR with depth implementation. The performance of the multi-sensor setup drops even more drastically during training not just in terms of time but also in other parameters. In Figure 5 the mean reward values are very inconsistent and the success rate is 0. This is because the drone often changes its path to avoid obstacles as it simply exceeds the maximum number of attempts to reach the goal per episode and the drone can't find any obstacle-free path to the goal. A very different

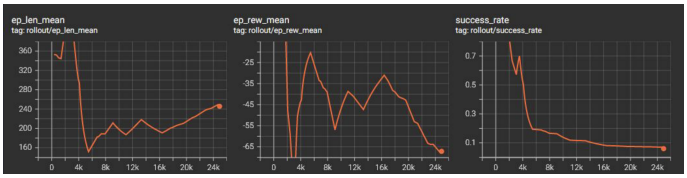

Fig. 6. Training metrics of implementation using LiDAR sensor only

story can be inferred from the above figure 6 as it has a decent success rate if not substantial and is more consistent and higher than the multi-sensor approach. However, it should also be noticed that in figure 5 the reward increases with timestamps unlike figure 6 where it is inconsistent. This is still not a huge positive as the range of value of the mean reward is still lower than a LiDAR-only setup.

B. Qualitative results

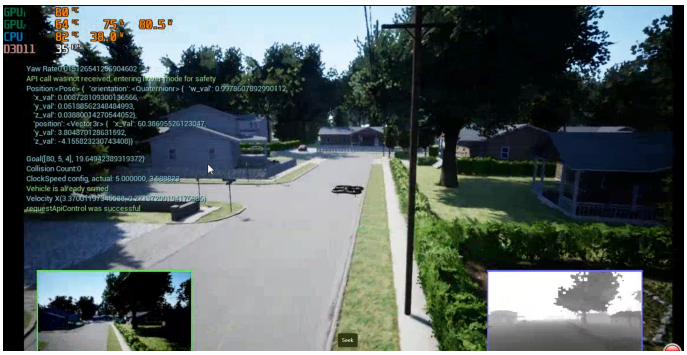

Fig. 7. Drone changing its course to avoid the pole in LiDAR-only setup

Qualitative results for both approaches showcase the advantages that LiDAR with depth sensor has over LiDAR-only implementation.In Figure 7 and the video of LiDAR-only Link) implementation shows that the drone in many episodes narrowly escapes the thin wooden pole and tree near which the goal is as the obstacle detection of the LiDAR is not very accurate. LiDAR and depth implementation video (Link) however showcases many instances when the drone actively tries to change its path to avoid obstacles. This is because the depth camera offers a more accurate and consistent obstacle detection.

However, the advantage comes with its downside of slow implementation as can be observed in the video(Link). This issue is persistent during training as well with Lidar with depth approach as can be seen from this video(Link), when comparing to LiDAR only training video(Link) this time gap is huge.

To understand better the obstacle avoidance capability of drones with LiDAR implementation a training video (Link) where the drone changes its course to avoid the pole can be used as a reference.

VI. Challenges

During this entire project, we encountered several challenges including the high training time of the model. Training for 100,000 time stamps took around 2 hours for the LiDAR and depth setup and 1 hour for the LiDAR-only setup. Although the time taken for training was manageable, AirSim consumed large amounts of RAM, affecting the workflow. Finding the correct LiDAR parameters and integrating them with the observation space was a significant challenge. When navigating in complicated environments, the training outputs were very random and was it was significantly harder to implement.

VII. Conclusion

This project provides a detailed comparison of two implementation methods for obstacle avoidance in drones using DDPG: LiDAR-only and LiDAR combined with depth sensors. While the LiDAR-only approach offers faster training times and consistent performance, the addition of a depth sensor significantly enhances obstacle recognition and avoidance. Despite the increased training and evaluation time, the improved accuracy of obstacle avoidance of the LiDAR and depth sensor combination is noteworthy.

Future work should focus on optimizing this integrated strategy to balance performance and efficiency effectively. There remains substantial scope for improvement in this project. Enhancements could include upgrading the algorithm from DDPG to TD3 for better results, improving generalization by assigning the drone multiple goals or waypoints and developing a more sophisticated reward function with parameterized weights (coefficients) for penalties and rewards to allow the model to determine customized weights for improved outcomes.

References

[1] AirsimDRL, "sunghoonhong/AirsimDRL/tree/master," 2022. Available: Link .
[2] Reinforcement-learning, "RahulSajnani/Reinforcement-learning," 2021. Available: Link
[3] DDPG-AirSim-Drone-Obstacle-Avoidance, "John-Venti/DDPG-AirSim-Drone-Obstacle-Avoidance," 2024. Available: Link

[4] H. Tan, "Reinforcement Learning with Deep Deterministic Policy Gradient," 2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA), Xi'an, China, 2021, pp. 82-85, doi: 10.1109/CAIBDA53561.2021.00025.

[5] A. Raffin, A. Hill, A. Gleave, et al., "Stable Baselines3," GitHub repository, 2020. [Online]. Available:Link.

[6] H. L. S. Dias, "UAV Navigation with Deep Reinforcement Learning in AirSim," GitHub repository, 2020. [Online]. Available: Link

[7] airsim2017fsr, author = Shital Shah and Debadeepta Dey and Chris Lovett and Ashish Kapoor, title = AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles, year = 2017, book title = Field and Service Robotics, eprint = arXiv:1705.05065, URL = Link

[8] fujimoto2018addressing, title=Addressing Function Approximation Error in Actor-Critic Methods, author=Scott Fujimoto and Herke van Hoof and David Meger, year=2018, eprint=1802.09477, archivePrefix=arXiv, primaryClass=cs.AI

[9] Shin, S.-Y.; Kang, Y.-W.; Kim, Y.-G. Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot. Appl. Sci. 2019, 9, 5571. https://doi.org/10.3390/app9245571

[10] @articlearticle, author = Tu, Guan-Ting and Juang, Jih-Gau, year = 2023, month = 01, pages = 57, title = UAV Path Planning and Obstacle Avoidance Based on Reinforcement Learning in 3D Environments, volume = 12, journal = Actuators, doi = 10.3390/act12020057

[11] 1606.01540, Author = Greg Brockman and Vicki Cheung and Ludwig Pettersson and Jonas Schneider and John Schulman and Jie Tang and Wojciech Zaremba, Title = OpenAI Gym, Year = 2016, Eprint = arXiv:1606.01540