applied sciences

MDPI

Article

iADA*-RL: Anytime Graph-Based Path Planning with Deep Reinforcement Learning for an Autonomous UAV

Aye Aye Maw [1], Maxim Tyan [1], Tuan Anh Nguyen [1] and Jae-Woo Lee [2,*]

[1] Konkuk Aerospace Design-Airworthiness Research Institute (KADA), Konkuk University, Seoul 05029, Korea; ayeayemaw@konkuk.ac.kr (A.A.M.); maxim@konkuk.ac.kr (M.T.); anhnt2407@konkuk.ac.kr (T.A.N.)
[2] Department of Aerospace Engineering, Konkuk University, Seoul 05029, Korea
* Correspondence: jwlee@konkuk.ac.kr

Abstract: Path planning algorithms are of paramount importance in guidance and collision systems to provide trustworthiness and safety for operations of autonomous unmanned aerial vehicles (UAV). Previous works showed different approaches mostly focusing on shortest path discovery without a sufficient consideration on local planning and collision avoidance. In this paper, we propose a hybrid path planning algorithm that uses an anytime graph-based path planning algorithm for global planning and deep reinforcement learning for local planning which applied for a real-time mission planning system of an autonomous UAV. In particular, we aim to achieve a highly autonomous UAV mission planning system that is adaptive to real-world environments consisting of both static and moving obstacles for collision avoidance capabilities. To achieve adaptive behavior for real-world problems, a simulator is required that can imitate real environments for learning. For this reason, the simulator must be sufficiently flexible to allow the UAV to learn about the environment and to adapt to real-world conditions. In our scheme, the UAV first learns about the environment via a simulator, and only then is it applied to the real-world. The proposed system is divided into two main parts: optimal flight path generation and collision avoidance. A hybrid path planning approach is developed by combining a graph-based path planning algorithm with a learning-based algorithm for local planning to allow the UAV to avoid a collision in real time. The global path planning problem is solved in the first stage using a novel anytime incremental search algorithm called improved Anytime Dynamic A* (iADA*). A reinforcement learning method is used to carry out local planning between waypoints, to avoid any obstacles within the environment. The developed hybrid path planning system was investigated and validated in an AirSim environment. A number of different simulations and experiments were performed using AirSim platform in order to demonstrate the effectiveness of the proposed system for an autonomous UAV. This study helps expand the existing research area in designing efficient and safe path planning algorithms for UAVs.

Keywords: reinforcement learning; anytime path planning

Citation: Maw, A.A.; Tyan, M.; Nguyen, T.A.; Lee, J.-W. iADA*-RL: Anytime Graph-Based Path Planning with Deep Reinforcement Learning for an Autonomous UAV. *Appl. Sci.* **2021**, *11*, 3948. https://doi.org/10.3390/app11093948

Received: 15 February 2021
Accepted: 22 April 2021
Published: 27 April 2021

1. Introduction

Real-time mission planning systems for autonomous vehicles such as UAVs have attracted widespread research interest over recent years. In these mission planning systems, path planning forms the main task, and dynamic path planning has been identified as one of the most important tasks for autonomous systems. To achieve a successful outcome from any mission, autonomous systems must have efficient planning mechanisms that are cost-effective, executable, and most importantly safe in terms of the UAV operation system. Many types of optimization, path planning, and machine learning algorithms have been proposed over the years to solve these problems for autonomous UAV systems [1–3]. However, the available literature shows that researchers have faced certain problems involving high computational times, insecure/inefficient collision avoidance, lack of intelligence,

and unreliable or non-optimum allocation of resources. The development of intelligent, computationally efficient planning is a topic that remains significantly challenging for current researchers, meaning that the development of an efficient algorithm that can find a high-quality solution for path planning and re-planning is necessary. Figure 1 illustrates the challenges that need to be overcome to create a successful real-time autonomous mission planning system for a single UAV system. The main challenges for real-time autonomous system are an efficient path planning and re-planning, ensure risk free for planning while avoiding collisions in the consideration of the UAV constraints such as velocity and/or maximum speed. Finally, the system should consider the safety in the environment which included not only static but also dynamic moving obstacles.

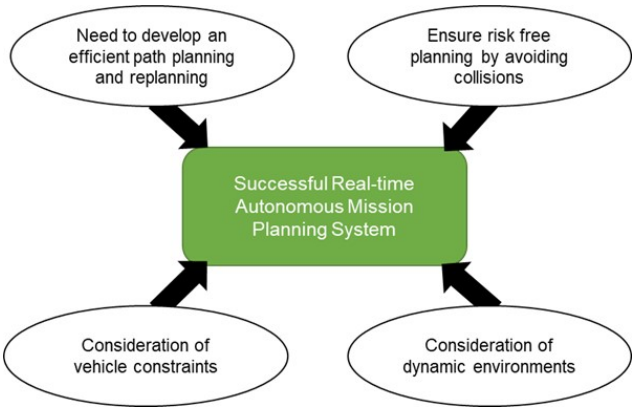


Figure 1. Challenges related to the development of a successful real-time autonomous mission planning system for a single UAV.

Numerous researchers have proposed schemes for real-time mission planning or path planning for mission planning system as a guidance and collision avoidance by combining a graph- or sampling-based path planning process with machine learning methods to create hybrid path planning approaches. These take advantage of a path planner to plan obstacle-free paths more rapidly, obtain a globally planned path, and exploit the benefits of reinforcement learning (RL) to allow the UAV to learn about the environment while traveling to the path in the local local destination while avoiding collisions. Most of the research in this area has focused on three degrees of freedom (3DOF) vehicles such as ground robots and cars. Huang et al. [4] proposed a system that aimed to achieve adaptive robot mission planning in a human traffic context, in environments crowded with people. This work focused on planning within a dynamic, complex environment by considering the constraints on UAV's velocity. The authors designed a mission planning system in which local navigation and operation were based on deep RL and global mission planning relied on a topological map. Park et al. [1] developed a novel, efficient, real-time hybrid path planning method by combining probabilistic road-map (PRM) path planning with RL to allow ground robots to deal with a dynamic environment. The objective of this method was to improve the adaptability of a PRM global planner in a dynamic environment. Faust et al. [5] proposed a hybrid path planning approach that combined PRM with RL for aerial cargo delivery tasks. In their approach, an RL agent first learned within an obstacle-free environment by employing a specifically designed feature vector to approximate the value function and to permit generalization beyond the training domain. Their method also considered the RL agent with continuous states as input and discrete actions as output. In addition, the research world could be extended to tasks in environments with static obstacles where the load–displacement must be bounded throughout the trajectory. The sampling-based motion planning is used to generate collision-free paths in the research. Faust et al. [6] extended their previous research in [5] to indoor planning with PRM-RL, using a point-to-point policy trained with DDPG as the local planner for PRM [7]. In the present paper, our algorithm creates a map from the sensors to control the velocity in the same way as the approaches developed by Faust et al., but our model is trained

with Auto Reinforcement Learning (AutoRL) method, which tunes the agent's reward and network architectures to improve the performance of the model and time needed for manual engineering. Dai et al. [8] applied meta-heuristics to the learning of the embedding network to solve the combinatorial optimization problem based on graphs. Due to the learned greedy policy behavior for the intended problems (i.e., minimum vertex cover, maximum cut, or TSP), the solution search within the network was more efficient than pointer network-based neural combinatorial optimization [3,9]. Francis et al. [10] also extended the work in [6] by proposing a hybrid path planning method that combined PRM with RL for long-range operation system for a ground robot in an indoor environment. In PRM-RL, an RL agent learns a local point-to-point task, incorporating system dynamics and sensor noise independent long-range environment structure. Chiang et al. [11] proposed a kinodynamic motion planning scheme based on the integration of a rapidly exploring randomized tree (RRT), and an efficient kinodynamic motion planner that used RL. The aim of this approach was to find an efficient solution to long-range path planning problems for ground robots. In [1,2,5,6,11], the proposed methods mainly involved static environments, and did not consider the vehicle constraints as in the RL-RRT approach [10]. The problems in this research is not taken into account in the re-planning phase. When a vehicle carries out planning in a real-world environment, initial path planning is not sufficient, and re-planning should be applied in order to minimize the failure rate, distance, and time. Basically, path planning algorithms which did not include the re-planning feature take a lot of computation time to generate the path in the real-time problem. For the real-world environment problems, the re-planning of the path is a necessary and critical issue for all of the algorithms. The path planning algorithm without considering re-planning can update the path again in real time, but the computation cost of the re-plan path is expensive. The algorithm must recalculate the whole path from the start to the goal destination again to get the obstacle-free path. For a novel anytime incremental search algorithm called improved the Anytime Dynamic A* (iADA*) algorithm, it considered the re-planning feature; in that case, if the obstacles or cost changes occurred in the initial path, the algorithm reused the previous planning information and updated the cost nearly in the affected area only. By doing these steps, the algorithm saves the computation time and can provide the new path very quickly.

The proposed research will bridge these gaps by developing an integrated planning system intended for the real-world environment. A hybrid path planning algorithm is developed by integrating a graph-based path planning algorithm with a learning-based algorithm for local planning to allow the UAV to avoid facing obstacles in real time. The global path planning problem is solved in the first stage using iADA*. A reinforcement learning method is used to carry out local planning between waypoints, to avoid any collision within the environment. This study aims to overcome a number of the significant challenges of mission planning as well as path planning in complex and dynamically changing environments for a UAV to reach the destination to achieve a successful mission. This research work aims to complete missions safely by doing path planning and re-planning together with avoiding obstacles in the dynamic environment. This work can be distinguished as one of the studies at a very early stage in the research area of hybrid path planning in which conventional path planning algorithms are integrated with deep reinforcement learning (DRL) for trustworthy guidance and collision avoidance of UAVs. In that regard, the main contributions of this study are as follows:

- Proposed a hybrid path planning algorithm in the integration of iADA* for global planning and deep reinforcement learning algorithms for local planning for UAV.
- Developed a simulator to mimic the realistic features of the real-world environment to achieve adaptive behaviors of a UAV for learning.
- Performed comprehensive evaluation and validation of the proposed hybrid path planning algorithm-based system in an AirSim simulation platform. The effectiveness of the proposed system for an autonomous UAV is investigated through simulations and experiments.

The outline of the paper is presented as follows: the motivation of the study and the related work are discussed in Section 1. The iADA*-RL path planning algorithm is proposed in Section 2. In Section 3, a case study on the implementation of the proposed path planning algorithm is presented with additional discussions on the future work. Conclusions are drawn in Section 4.

2. Proposed iADA*-RL Planning Algorithm

2.1. Overall Description

Path planning, re-planning, and collision avoidance in real time are essential mission management tasks for an autonomous UAV, and proposed solutions need to account for both static and dynamic obstacles. A few of these obstacles are known a priori, and the research presented will therefore follow a classic two-layer architecture. In the first stage, the global planning module, which assumes the existence of a fully or partially known static environment, determines and generates an obstacle-free path from a given starting location to the target (known as the destination location) to achieve the mission. This task is performed before the UAV executes the mission task. The UAV will follow this initial path, and, as its sensors detect a new obstacle or threat, the local planning module is engaged to avoid it and provide a safe route for global path planning. The global path planning algorithm must then re-plan the path from the location provided by the local planning module. Figure 2 represents a mission scenario in which the global path needs to be adjusted to avoid both static and dynamic obstacles when a UAV is flying within a given environment. The data flow and architecture of the proposed iADA*-RL algorithm can be seen in Figure 3. The architecture included three main modules such as global planning and re-planning, learning-based local planning while avoiding obstacles in the execution of the mission and the real-time system monitoring.

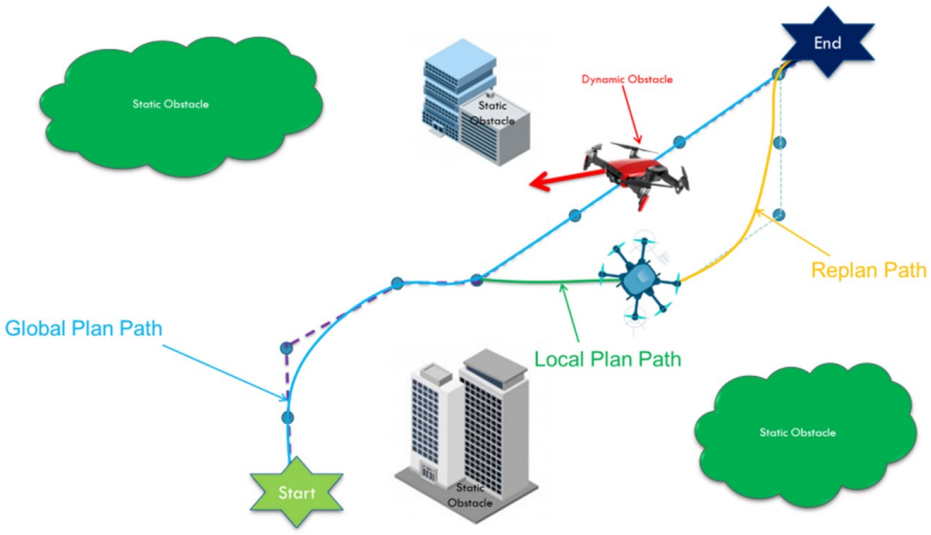


Figure 2. Real-time mission of a UAV with globally and locally planned paths.

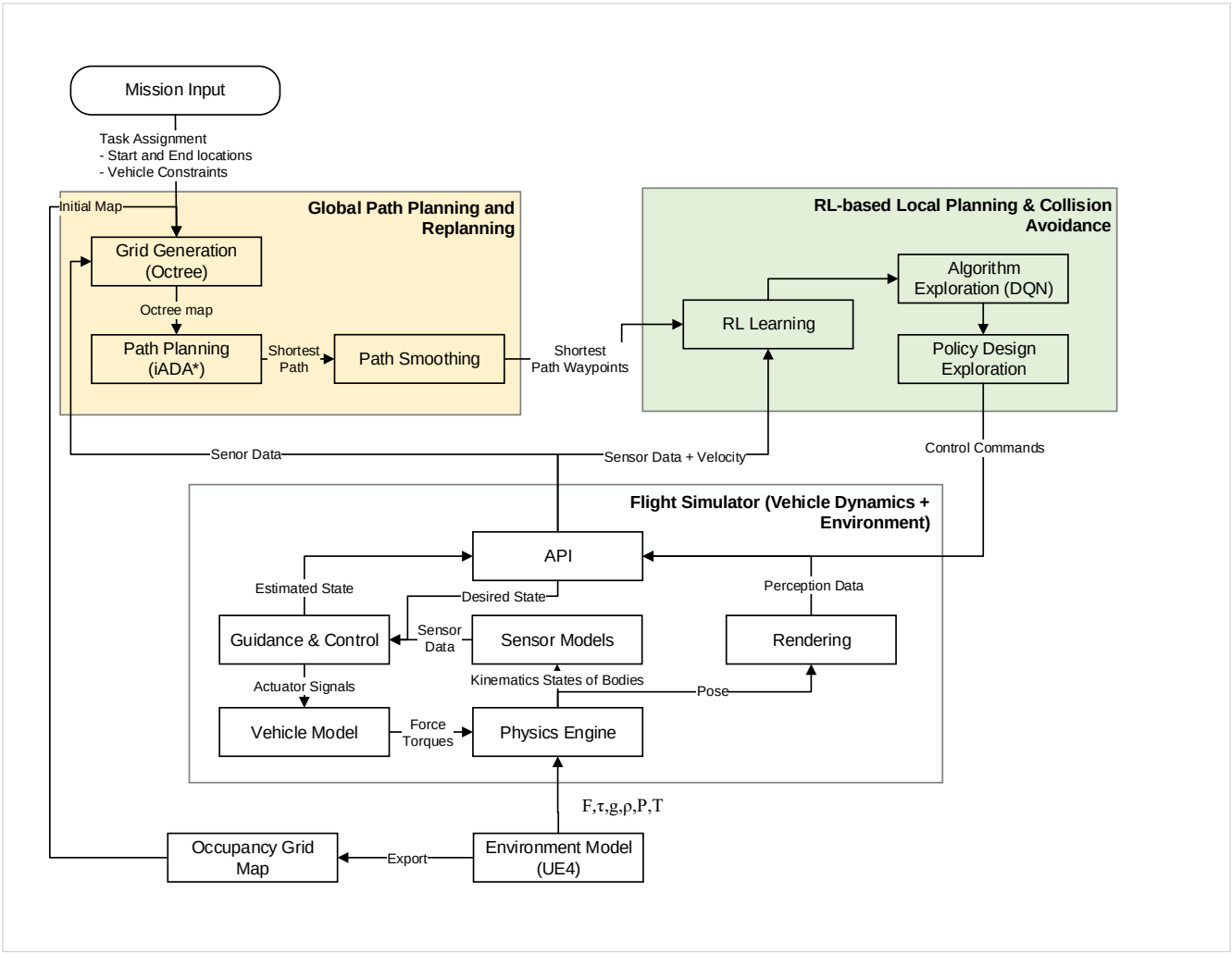


Figure 3. Proposed architecture of the hybrid path planning system.

2.2. Global Path Planning and Re-Planning iADA Algorithm*

The first stage in this research involves global path planning and a re-planning focus on developing an efficient path planning algorithm iADA* [12] for a single UAV system. The iADA* algorithm is based on anytime path planning algorithms and can improve the calculation of the path lengths and decrease the frequency with which the path needs to be calculated throughout the search, which makes the algorithm significantly faster. Furthermore, it has a simultaneous re-planning capacity to avoid obstacles, which also reduces the computational time. The iADA* algorithm improves the calculation of the path lengths and decreases the calculating frequency of the path throughout the search, making it significantly faster than the existing anytime dynamic A* (ADA*) algorithm. The algorithm is designed to provide an efficient solution to a complex, dynamic search environment when the locally changes are affected.

One of the main features of the iADA* algorithm is allowing the state recalculation during the iteration after the initial path was generated. When there are changes affecting the environment, the algorithm uses a search method for rearranging the states list to reduce the frequency of calculating the path throughout the search. This step makes it significantly faster compared with current algorithms for solving concurrent path planning and re-planning problems. Additionally, new constraints are introduced via creating a virtual wall, in order to prevent the UAV from traveling across complicated obstacles, such as a U-shaped obstacle. Moreover, if necessary, the UAV will not move again on the same grid. The constraint also protects the path from traversing between two obstacles,

especially when the obstacles are facing each other. The new method's improvements over the original ADA* include the method of cost calculation, search time, and total time to arrive at the goal. The iADA* has an improved feature compared with the current path planning algorithms such as D*Lite, D*Extra Lite, MPGAA*, and ADA* algorithms [12]. Path planning on a random map has demonstrateda successful solution of concurrent path planning and re-planning problems. From the data obtained, iADA* provides an optimum path for the UAV with faster calculation. This algorithm is implemented in the global path planning and re-planning module. The first module provides a pre-planned path that includes a set of waypoints for the initial stage of the mission, a database of fully or partially known static obstacles, and a terrain map. If obstacles are found within the segments that connect the given waypoints, an obstacle-free path that satisfies certain specific optimization criteria, such as a minimum distance, is provided in the form of a waypoint matrix. During execution of the mission, the path following module is responsible for following the pre-planned path. As the path is executed, the RMP system provides current sensor data, and the position and velocity of the UAV to the local planning and collision avoidance module. Pseudo-code for the iADA* path planning algorithm is given in Algorithm 1.

Algorithm 1: *iADA**

```
Begin
    Function Key(s):
        if (g(s) > v(s)) then
            return [v(s) + ε · h(s_start, s); v(s)]
        end
        else
            return [g(s) + h(s_sart, s); g(s)]
        end

    Function UpdateState(s):
        if (s was not visited before) then
            g(s) ← ∞
        end
        if (s ≠ s_goal) then
            v(s) ← min_{s'∈Succ(s)} (c(s, s') + g(s'))
        end
        if (s ∈ OPEN) then
            remove s from OPEN
        end
        if (g(s) ≠ v(s)) then
            if (ε = ε_0) then
                if (s' ∉ CLOSED) then
                    insert or update s' in OPEN with Key(s)
                end
                else
                    insert s' into INCONS with Key(s)
                end
            end
            else
                if (s' ∉ CLOSED) then
                    insert or update s' in OPEN with Key(s)
                end
                else
                    remove s' from CLOSED to OPEN
                end
            end
        end
```

Algorithm 1: *Cont.*

```
39
40:  Function ComputeShortestPath(s):
41:      while (min_{s∈ OPEN ( key (s))} < key(s_start ) OR rhs (s_start ) > g(s_start )) do
42:          s ← state with the minimum key from OPEN
43:          k_old ← key(s)
44:          k_new ← CalculateKey(s)
45:          if s′ of s only one and s′ = Succ(s) then
46:              insert state s into VWALLS
47:              mark state s as a temporary obstacle
48:          if k_old > k_new then
49:              insert s into OPEN with k_new
50:          else
51:              if g(s) > v(s) then
52:                  g(s) ← v(s)
53:                  remove state s from OPEN
54:                  CLOSED ← CLOSED ∪{s}
55:                  for all s′ ∈ Pred (s) do
56:                      UpdateState(s′)
57:              else
58:                  g(s) ← ∞
59:                  for all s′ ∈ Pred(s) ∪ {s} do
60:                      UpdateState(s′)
61:  Main():
62:      g(s_goal ) ← ∞; ε ← ε_0; s_goal ← s_start
63:      for all s ∈ S do
64:          g(s) ← ∞
65:          v(s) ← ∞
66:      v(s_goal ) ← 0
67:      OPEN ← ∅
68:      CLOSED ← ∅
69:      INCONS ← ∅
70:      VWALLS ← ∅
71:      insert S_goal into OPEN with key(s_g oal )
72:      ComputeShortestPath(s)
73:      publish ε - current suboptimal solution
74:      while S_start ≠ S_goal do
75:          s_start = arg min_{s′∈Succ}(s_start )(c(s_start , s′) + g(s′))
76:          move to S_start
77:          scan graph for changed edge cost
78:          if changed in edge costs are detected then
79:              ε ← ε + Δε
80:              remove all states in VWALLS from temporary obstacles
81:              VWALLS ← ∅
82:              for all directed edges (u,v) with changed edge costs do
83:                  c_old = c(u, v)
84:                  update the edge costs c(u, v)
85:                  if (c_old > c(u, v)) then
86:                      rhs(u) = min(v(u), c(u, v) + g(u))
87:                  else
88:                      if (v(u) = c_old + g(u)) then
89:                          if (u ≠ s_goal ) then
90:                              v(u) = min_{s′∈Succ(u)}(c(u, s′) + g(s′))
91:              UpdateState(u)
92:          else
93:              if ε > 1 then
94:                  ε = ε − Δε
95:          if INCONS ≠ ∅) then
96:              move state s from INCONS into OPEN
97:          update the priorities for all s ∈ OPEN according to key(s)
98:          CLOSED ← ∅
99:          ComputeShortestPath(s)
100:         return publish ε− current optimal solution
101:         if ε = 1 then
102:             wait for changes in edge cost
```

2.3. Vehicle Dynamics

The implementation of an autonomous UAV system requires a vehicle dynamics model. A vehicle dynamic model used in this research was developed by the Microsoft research team, called AirSim [13,13]. This is a simulator for UAVs and cars, and can be integrated with any Unreal Engine 4 (UE4) [14] or Unity game engine [15]. It is open-source, cross-platform, and supports hardware-in-the-loop (HITL) with popular flight controllers such as PX4 to create physically and visually realistic simulations. The core components of AirSim include the quad-copter model, sensor model, physics engine, rendering engine, environment model, and application programming interface (API) layer [16]. As a flight controller, AirSim supports PX4 [17,18] for HITL simulation [19], name 'Simple Flight' controller as the default and also supports Ardupilot firmware for software-in-the-loop (SITL) simulation [20]. The Simple Flight controller can control the UAV by taking in desired input such as the angle rate, angle size, velocity, or position. Each axis of control can be specified based on one of these modes. Internally, Simple Flight uses a cascade of proportional-integral-derivative (PID) controllers [21] to generate the actuator signals. This means that the position PID drives the velocity PID, which in turn drives the angle level PID, which finally drives the angle rate PID. For example, in the case of a quad-rotor, the user can specify certain pitch, roll, and yaw angles as the desired state, and the flight controller can then use sensor data from the accelerometer and gyroscope to estimate the current state and then to compute the motor signals needed to achieve the desired state. The rendering engine translates these calculations of the position and orientation into a visual simulation, and the API layer then receives sensor and perception data, such as a camera image, Light Detection and Ranging (LiDAR) image, etc.

2.4. Learning Algorithm for Local Planning

2.4.1. RL-Based Local Planner

Nowadays, many researchers investigated and proposed real-time mission planning or path planning for guidance and collision avoidance problems by combining a graph or sampling-based path planning and machine learning methods as a hybrid planner, by taking advantage of the path planner to plan the obstacle free path faster, get the global planned path, and use RL's advantage to learn about the environment while traveling to the local destination and avoiding facing obstacles. Most of the research focused on 3DOF vehicles such as ground robots and cars.

With inspiration from the works [6,11,22], this research proposes DRL-based local planning and collision avoidance for UAV in the dynamic environment. In the current research, the local planner relies on discretized actions to produce the local path and avoidance action. This research mainly focuses on the development of the Deep Q-Learning based local planning and collision avoidance. For the research and experimental purposes, we also implement the Deep Deterministic Policy Gradient (DDPG) network and integrate it with iADA* path planning. DDPG is a more advanced DRL method, which combines ideas from Deterministic Policy Gradient (DPG) and Deep Q-Network (DQN). Hence, this research also makes a comparison between iADA*-DQN and iADA*-DDPG.

2.4.2. Problem Statement

The main elements of an RL problem involve an agent, environmental states, a policy, a reward function, and a value function [23–25]. A neural network is utilized as a nonlinear function estimator to estimate the action-value function, which improves the RL process and gives a DRL problem. The proposed collision avoidance method is based on DRL. The collision avoidance problem can be defined as an Markov decision process (MDP) [26], in which a tuple (S, A, P, R, y) is used to represent an MDP where all possible states s are included in the state space $S(s \in S)$. An action space A contains the possible actions $a(a \in A)$, the state transition model is denoted as M, the reward function is R, and a discount factor γ is applied. A neural network with two hidden layers is used as a nonlinear function approximator to estimate the action-value function $Q(s, a; \theta)$ with weights θ.

A vast amount of interaction data, such as the transition, needs to be collected to train the neural network. AirSim is used to acquire these data and to train the neural network at each time step t, the UAV sends the current state s_t to the neural network as input, and the output is the action value for the current state. The UAV then needs to choose an action based on certain policies, such as an ε—greedy policy in which the value of ε decreases from one with a decay rate of β, as follows:

$$\varepsilon_{k+1} = \beta\varepsilon_k \quad (1)$$

where k is the epoch, and ε will stop decaying and remain fixed when it reaches a value of 0.05 after carrying out the chosen action a_t, the UAV is now in a new state s_{t+1} and receives a reward r_{t+1}. All of the transitions are recorded in order to update the network.

2.4.3. Simulator Development

DRL requires a large input dataset in order to generate appropriate behaviors. In particular, collision avoidance behaviors must be acquired through learning. To learn these avoidance behaviors, a series of control command actions that cause collisions with obstacles need to be repeated. These actions cannot be performed in the real-world, and we therefore aim to train the autonomous mission planning system for the UAV on a simulator and then to apply the results of this training stage to the actual autonomous mission planning system.

There are many popular simulators for drones or UAVs, including AirSim, Gazebo [27] and V-REP [22]. We used AirSim in our experiments, as this was developed as an Unreal plugin that can easily be dropped into any Unreal or Unity environment. AirSim also provides high-fidelity simulations and dynamics for UAVs. The built-in simulation model in AirSim is Parrot AR Drone 2.0 [28]. Due to the use of UE4, AirSim can take advantage of recent advancements in rendering techniques in the game industry.

UE4 provides state-of-the-art graphic features, such as photo-metric lights, planar reflections, physically based materials, ray-traced distance field shadows, and numerous camera angles. It is also an open-source program, meaning that it is easy to emulate a range of real-world scenarios, for example setting up a number of both static and dynamic obstacles with different speeds [29]. AirSim also provides dynamic simulations of both the UAVs themselves and their sensors and actuators. Embedding a UAV model into an environment in a real-time simulation can be done by deploying a physics engine running at 1000 Hz. Figures 4 and 5 show examples of static and dynamic simulation environments for the UAV, taken from the airlearing-ue4 project [30]. The UAV is not only expected to avoid obstacles in real time but also to follow the waypoints generated by the global path planner.

Figure 4. Screenshots of the block (static) environment. [31]

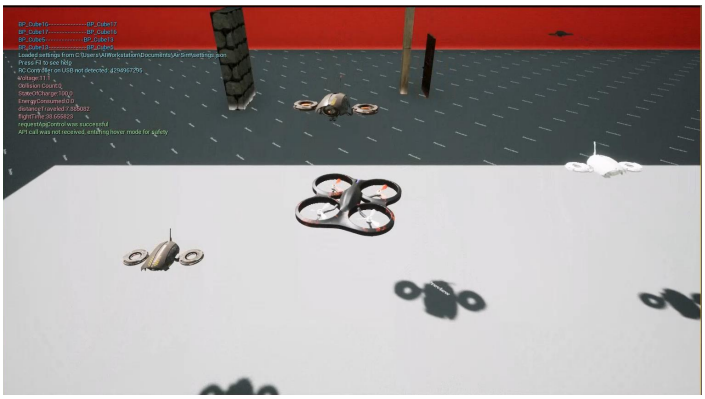


Figure 5. Environment used to test for dynamic obstacle avoidance [30].

2.5. RL Agent Training

The network architecture for the DRL agent, for both DQN and DDPG, used in this experiment is shown in Figure 6. The network takes a 30 × 100 state representation, current position, and velocity as inputs. The first layer convolves 32 kernels of 8 × 8 with stride three with the input image, with an Exponential Linear Unit (ELU) value of 0.8, and uses a Rectified Linear Unit (ReLU) activation function. The second convolutional layer obtains 64 kernels of 4 × 4 of stride two and an ELU value of 0.8, and is again followed by a rectifier. The third convolutional layer convolves 64 kernels of 3 × 3 of stride one, followed by a rectifier, while the output from the fourth layer is flattened and concatenated with the position vector. The combined inputs are fed to the two fully connected layers, where the first layer includes 256 hidden units and the second 128 hidden units.

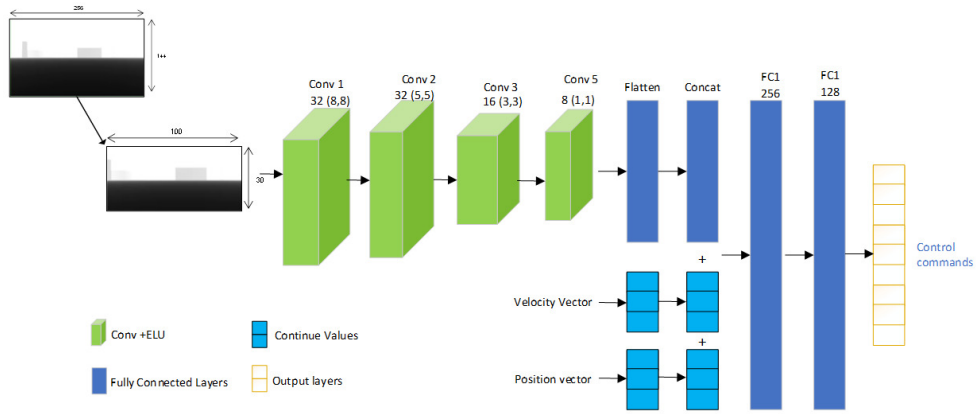


Figure 6. Network architecture for the DQN and DDPG agents.

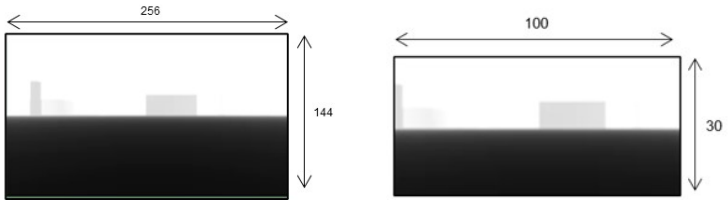
The action space determines the number of hidden units in the last fully connected layer. In our experiment, we used eight action spaces.

Action space: The action space is a set of all actions that the agent can act out in a certain environment. It consists of eight discrete actions that can be taken to avoid obstacles. One involves moving in the direction of the current heading, with a speed of 1, 2, 3, or 4 m/s, and another involves moving backward along the current direction with a speed of 1 m/s. A further two actions involve rotating to the right and to the left directions by 30 degrees. The remaining two actions involve moving to up and down positions, respectively, with a speed of 2 m/s. At each time step, the policy is used to choose the observation space as the input, and the output is one of these eight actions. These high-level actions are mapped to the low-level flight command using the flight controller provided by AirSim.

State space: The state space is a set of all the states that the agent can transition to and use in the training and learning process of the agent. The state of the input used for the

input layer is configured using (i) depth image data; (ii) relative goal position, and current velocity as follows.

1. Depth image data: The state representation must provide the information needed by the agent to avoid real-time obstacles. To avoid these obstacles, the agent receives a pre-processed depth image from the drone's front camera, and follows and returns to the planned path and waypoints provided by the iADA* global path planner. The agent also receives the current position and state of the drone. The depth image shown in Figure 7 shows the distances to the surfaces of the obstacles seen by the camera, mapped to grey-scale, in an image of size 256 × 144 pixels. In this experiment, we applied a basic image processing step to reduce the dimensionality of the input, in order to reduce the computation and memory requirements. A drone is typically moving along a fixed plane, and the middle section of the image is therefore most relevant. The size of the pre-processed image used is 100 × 30 pixel sizes.



(**a**) Depth image (**b**) State representation

Figure 7. Representation of states in the depth image and pre-processed image.

2. Relative goal position and current velocity: The relative global position is a 3D vector that represents the goal in polar coordinates (distance and goal) with respect to the current position of the UAV vehicle. The distance s_{distance} and the direction to the destination are calculated using Equation (2):

$$s_{\text{distance}} = \|\rho\| = \left\| P_{\text{curent}} - P_{\text{goal}} \right\| \tag{2}$$

Finally, the reward used in the learning process is set using Equation (3), following the approach used in [2]. When the UAV reaches the destination, an arrival reward is given. In this study, the condition $(s_{\text{distance}} < 0.5)$ is regarded as representing arrival at the destination. When colliding with obstacles such as walls, other UAVs or cars, a collision reward is given. In other cases, a positive or negative reward is given based on the relationship between the UAV and the destination. A positive reward is given for actions that approach the destination, and a negative reward for actions that move away from it. $s_{\text{distance }(t)}$ represents the value of s_{distance} at time t.c_r is an attenuating hyper-parameter:

$$r = \begin{cases} 50 & \text{arrived} \quad \text{if } (s_{\text{distance}} < 0.5) \\ -50 & \text{collision} \\ \left(s_{\text{distance }(t-1)} - s_{\text{dis tance }(t)}\right) \cdot c_r & \text{else} \end{cases} \tag{3}$$

If the drone bumps into an obstacles, either static or dynamic, the agent will receive a reward of -50. A new epoch starts with the drone in the initial position of $[0, 0, 0]$, which indicates the North, East, Down (NED) point coordinate in the AirSim simulator. To update the weights of the network, we apply a mini-batch gradient descent to the loss function, as follows:

$$L(\theta) = \frac{1}{2n} \sum (y_i - Q(s_i, a_i; \theta))^2 \tag{4}$$

where y_i are the current target outputs of the action-value function, and n is the size of the mini-batch. To avoid overestimation, the agent updates the target y_i using the following equation:

$$y_i = r_{i+1} = \gamma Q(s_{i+1}, a; \theta); \theta^{-})) \quad (5)$$

where r_{i+1} is the immediate reward after taking action a_i, and γ is the discount factor. Another neural network $Q(s, a; \theta)$ is initialized with random weights θ^{-}. This network chooses the control command action with maximum values Q', and the value of the action a decided in the target network is chosen from the neural network Q', but the value of a is decided in the target network.

2.6. Learning Phase

The proposed autonomous mission planning system for an UAV combines a global path planner based on an iADA* graph with local planning based on learning between the nodes of the global path planner. The global planner and the local behaviors acquired through learning must be prepared in advance, and the system provides initial learning results in order to acquire these local behaviors. In this section, the training flow is summarized and the result of the training network is also presented. For the training phase, a series of processes from action selection to learning rates are repeated. This series of proposes is called *'step'*. An 'episode' refers to a series of UAV steps from the initial position to the destination position, or a position at which the UAV hits obstacles. In the training processes, episodes including several steps are looped. The integration of the initial start and goal positions of the UAV are randomly selected at the beginning of each episode. In the training process, the environment also including both static and dynamic moving obstacles. All the dynamic moving obstacles are moving with different speeds and directions randomly. In the action output selection, the index number of actions i is obtained on ϵ-greedy algorithm. An ϵ-greedy selects an action that takes the maximum Q-value in the current state s_t or a random action with a higher probability at the beginning of the learning process. As a result of action $a_t(i)$, reward r_t and judgment d_t and the next UAV's state $s_{t=1}$ are obtained. Then, these are added to experience memory. Learning is performed by retrieving randomly from the experience memory and updating the unit weights of the neural network by the error back propagation algorithm. From these processes, the Q-value of the action which should take in the s_t is updated to become higher. In this process, the UAV learns the behaviors needed to move towards the destination by avoiding obstacles along the path. The destinations for the learning stages were set randomly. The training process in the experiment took around four days to finish the training. Figures 8 and 9 show the accumulated rewards and losses applied during the training of the DQN network, and it can be seen that the rewards change from -150 to 120 after 200,000 steps. This is a way to check that the agents effectively get better. In the current training result, the accumulated reward of the Q-learning does not finish as a convergence when the experiment stops the training, and we stopped the training after we received our goal reward value 100. The goal for the reward value is decided when the mission success cases achieve more than 70% of running the experiments test cases in the simulation. This means that the UAV can learn how to arrive at the destination and to avoid dynamic obstacles within a low number of training steps. In Figure 8, the orange line represents the moving average trend-line of the DQN model.

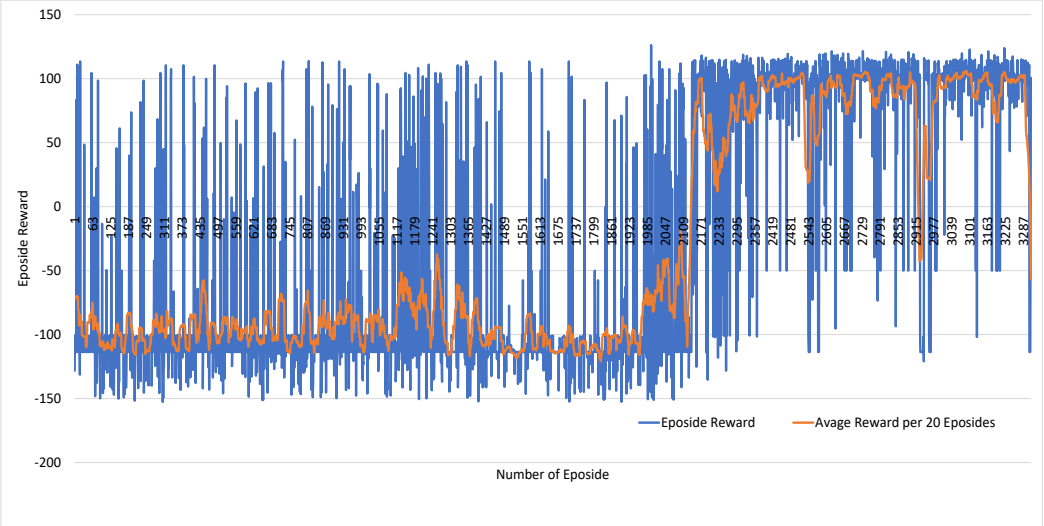


Figure 8. Training: ϵ-greedy policy of DQN model.

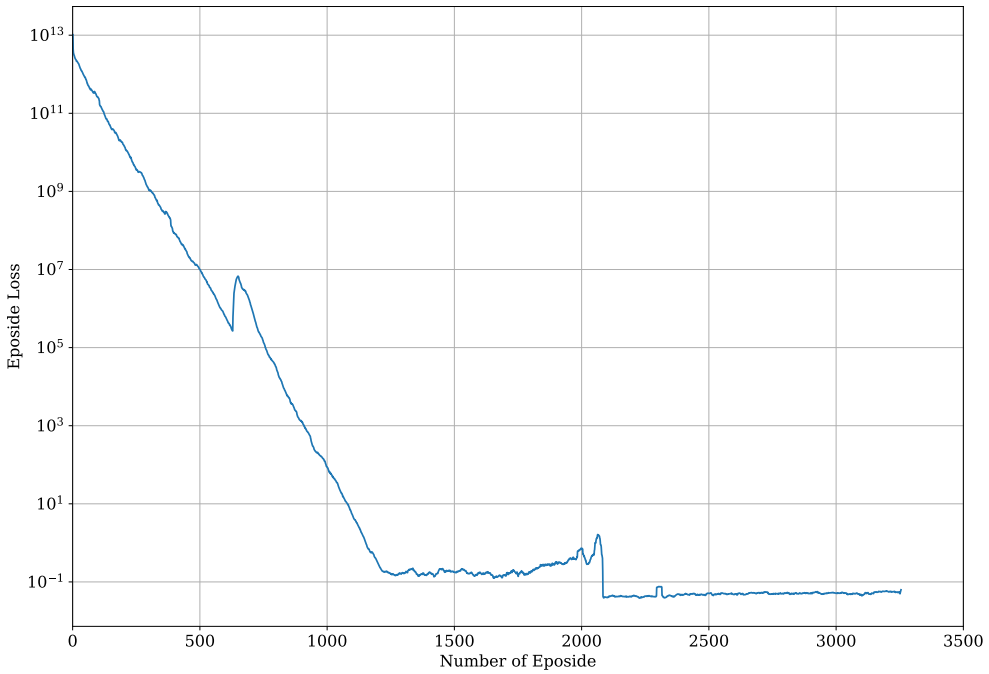


Figure 9. Loss values for the DQN model.

3. Case Study: Mission Planning for an Autonomous UAV

3.1. Experimental Setting in Simulation

Our experiments were carried out using two computers, one of which acted as a companion computer and one as a host. The companion computer is a powerful compute unit (compared to the flight controller) that is responsible for the processing of the high level, computationally intensive tasks such as vision processing. The architecture of the experimental system setting in the simulation is shown in Figure 10. The aim of the experiment run on the companion computer was to test the behavior of a real drone. This computer was responsible for running the path planner and for generating the octree [32,33], for which we used the Robotics Operating System (ROS) running on Ubuntu 16.04. The iADA* path planner algorithm was implemented in C++. The specifications for the companion computer were as follows: Intel Core i7-7700, 16 GB RAM, and Nvidia GTX 2070 GPU. This computer was responsible for running the collision avoidance DRL algorithms, while the AirSim simulator and environment were run on the host computer.

The DRL algorithm was implemented in Python using the Keras-rl library and the hardware specifications of the host computer were as follows: Intel Core 17 (10th generation), 16 GB RAM, and Nvidia GTX 2070 Super GPU.

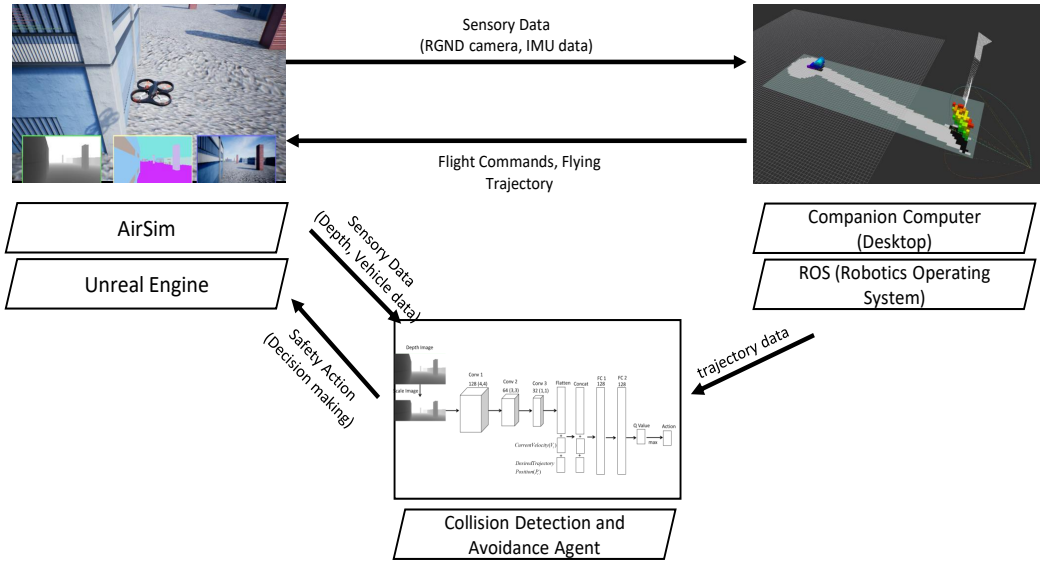


Figure 10. Architecture of the SITL system.

The overall system was tested, and the results were evaluated based on the success and failure rates of the system, in static (block) and dynamic environments, with the same settings for the implementation of DQN. The size of the static environment was 300 × 200 m, while that of the dynamic environment was 100 × 100 m, with no limitations on altitude. The dynamic environment was restricted to a small, indoor area because the focus was on avoiding very complex dynamic obstacles. There were 17 obstacles of different sizes, seven of which were static and the remaining 10 dynamic. The obstacles moved randomly within the environment with velocities ranging between 5 and 25 m/s, with no limitations on the possible directions.

3.2. Experiments and Results

The overall system was tested based on 50 test cases for both environments. A randomized mission was assigned to the UAV, and as it flew along the planned path generated by the iADA* path planner, the DQN agent continually checked for dynamic obstacles. If the DQN detected a dynamic obstacle along the path, the agent generated an action behavior to safely avoid the obstacle. When the UAV had arrived safely at the waypoint, the iADA* path planner re-planned the path to arrive at the overall goal of the mission. This procedure was applied only to dynamic obstacles; unknown static obstacles were detected using the octree, and the octree map was regenerated to allow the path planner to re-plan the path.

In this research, experiments were considered into two cases for the mission task cases: success and failure cases. Success cases mean when the UAV will start from the start destination and arrive the goal destination successfully without attacking any obstacle on the way, this case considered as a completed mission. Otherwise, it was failure cases which considered that the UAV could not finish its assigned mission. The case which is considered as a failure case is divided into two cases. For the first case, the algorithm could not find a path between the start and the target destination. In that case, the UAV could not start because there is no path to follow and fail to complete the mission. These kinds of constraints can see if the environment is a GPS denied environment so the goal destination can be on the obstacles or no fly zone in the reality. Another case with consideration as a failure is when the UAV starts the mission and hits the obstacle on the way before it finishes the mission.

For the static environment, the success rate is 94%, and, for a total of 50 test cases, there are only three instances of failure when running only the iADA* global path planning and re-planning algorithms in known static environments, as shown in Table 1. In the experimental test cases, the start and goal destinations of the UAV mission are randomly assigned. Failures occurred because the algorithm could not find a path between the start and the target points. The obstacle hitting failure case is not observed because the global path planning and re-planning algorithm is already aware of the environmental information beforehand, and the path planned by the algorithm had already considered the static obstacles. The three cases of failure are eliminated when iADA* is used with DQN and DDPG, since the local planners can travel to the nearest goal point if there is no goal point; hence, although the path is not optimal, there is no failure, and the mission could be achieved successfully.

The path planning for the UAV may fail within the dynamic environment, but the success rate can be acceptable, with a value of over 90%. Table 2 shows an analysis of the results from three methods: the iADA* algorithm alone, iADA* with DQN, and iADA* with DDPG. In total, we evaluated 50 test cases with 12 dynamic obstacles and 10 static obstacles, over an area of 100 × 100 m. For the iADA* algorithm alone, we found 13 cases of failure. The failures are mostly that the UAV is attacked by the obstacles because the iADA* algorithm creates an octree grid using octomap in real time, but, due to the computation time required to create the grid map, the UAV could not avoid moving obstacles and hitting the obstacles. Some small cases of this failure occurred because the UAV could not find a path between the start and goal point, and this is mostly due to the presence of a dynamic obstacle along the path. When a failure is observed due to an inability to find a path, this arose due to the random generation of start and goal locations, in which some points are co-located with obstacles, meaning that the path planner could not find them. As a result, the overall success rate is 74% for 50 test cases, and hence the failure rate is rather high for this low number of test cases.

Table 1. Comparison of success rates for different approaches for 50 test cases in a static environment.

	Algorithms		
	iADA *	**iADA * + DQN**	**iADA * + DDPG**
Success	47	50	50
Failure	3	–	–
Success rate	94%	100%	100%

Table 2. Comparison of success rates for different approaches for 50 test cases in a dynamic environment.

Task	**Algorithms**		
	iADA *	**iADA * + DQN**	**iADA * + DDPG**
Success	37	47	45
Failure	13	3	5
Success rate	74%	94%	90%

3.3. Discussions and Future Work

Using a complete framework, we ran experiments with a minimum of 50 test cases and analyzed the results in terms of the success rates for static and dynamic environments, using three different methods: iADA* alone, iADA* with DQN, and iADA* with DDPG When iADA* was implemented with DQN, the results were fairly good, with a total success rate of 94% and only three cases of failure. A summary of the results for the dynamic obstacles involved environment can be seen in Table 2. The high success rate was attained because DQN is a local path planner, and the avoidance of collisions allowed the iADA*

algorithm to find paths more easily. In this approach, each case of failure was due to a collision with a dynamic obstacle. Failure arose because, when the collisions were moving in a direction in which the UAV was not heading, the UAV's sensor could not identify them. Finally, when iADA* was implemented with DDPG, the results were also fairly good, with a success rate of 90% and five cases of failure. All of these failures were due to unavoidable dynamic obstacles. This approach therefore achieved a better result than iADA* alone but was less effective than when DQN was used. This was because the controller used for the simulator also needs to be considered. DQN is more effective when the system has a low-level controller in comparison with DDPG, and this may be the reason that DQN outperformed DDPG. By comparing one of the highest results and recent papers from the research which used PRM-RL [10] applied for the ground robot, in the dynamic obstacle cases, it obtained 82% accuracy per 20 test cases and, for this current paper's research, the experiment achieved a success rate of greater than 90% in the 50 test cases. Based on this fact, for the iADA*-RL algorithm based on iADA* with DQN and DDPG learning, the results were acceptable for a complex, dynamic indoor environment. The results from RL learning also depend on the parameter setup and training time, and the results presented here could be improved by setting up the parameters carefully and enhancing the currently trained model.

Simulations and experiments were carried out to test our scheme using a real-time simulation platform, and it was shown that the proposed mission planning system was effective in an environment with complex obstacles. The autonomous mission planning system was successfully implemented and tested and was shown to avoid obstacles in a real-time simulation environment. Since we have achieved many successes and improvements on this research problem in the simulation environment, we could extend our proposed research system in the real-world environment with practical UAVs. For the future work of this research, there are still several problems that remain with our approach. Firstly, the collision avoidance scheme for moving objects needs to be improved to include not only a single sensor but also a 360°-view sensor. For the moving obstacles, the environment used here for testing contained dynamic obstacles that moved with constant directions and speeds and should be updated to consider more realistic objects. The results from our approach still need to be validated using an actual UAV system. For the path planning and re-planning task, basically if the system gets the exact information of the environment map, then the optimal path can be obtained.

To ensure the safety of the UAV, we will implement three steps to check the accuracy of the collision detection and avoidance task. In the first step, we implement the collision detection and avoidance system in the simulation. This step was already implemented in this paper's experiment results. For the second step, the system implements the collision detection in the practical UAV and checks the detection accuracy. If these steps provide acceptable, high accuracy results, move to the final step. If the collision detection task doesn't achieve 100% accuracy, then we will enhance the DRL network by learning, using the real-world dataset to improve the accuracy. In the final step, the system will implement the collision detection and avoidance system in the practical UAV. In this way, the system can improve the safety of the UAV by avoiding collisions in practical situations. Additionally, our proposed methodology can be applicable in the practical UAV systems, such as the transportation system proposed in [34]. Therefore, our proposed hybrid path planning is feasible in such drone-based transportation systems, and this is a fruitful research avenue.

4. Conclusions

This study has proposed an autonomous mission planning system for an autonomous UAV system, with local planning based on DRL and global planning based on an iADA* algorithm. The vehicleUAV learned in an AirSim simulation environment that included both static and dynamic obstacles, and generated policies for traveling to local destinations while avoiding upcoming obstacles. The system also considered the vehicleUAV dynamics and parameters such as the speed and acceleration of the UAV. A global path planner

was used to calculate the sub-optimal path and to re-plan the path if necessary. The experiment was conducted with a minimum of 50 test cases, and the results were analyzed in terms of the success rates for static and dynamic environments, using three different methods: iADA* alone, iADA* with DQN, and iADA* with DDPG. In a static environment, the success rate was 94% for the test cases when iADA* was used alone as a global and local planner. When iADA* was applied with either of the two learning methods as a local planner, no cases of failure were observed. A dynamic environment was also tested using these three methods, with 50 test cases. The success rate was 74% for the iADA* algorithm alone, 94% for iADA* with DQN, and 90% for iADA* with DDPG. The analysis results showed that our real-time autonomous mission planning system achieved the best performance when the iADA* algorithm was used for global path planning and re-planning, and DQN was used as the local planning and collision avoidance algorithm.

Author Contributions: Conceptualization, A.A.M., M.T., and J.-W.L.; methodology, A.A.M.; software, A.A.M.; validation, A.A.M., M.T., and J.-W.L.; formal analysis, A.A.M., M.T., and J.-W.L.; investigation, A.A.M., M.T., and J.-W.L.; resources, A.A.M.; data curation, A.A.M.; writing—original draft preparation, A.A.M., M.T., and T.A.N.; writing—review and editing, A.A.M., M.T., T.A.N., and J.-W.L.; visualization, A.A.M.; supervision, M.T., and J.-W.L.; project administration, J.-W.L.; funding acquisition, J.-W.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Korea Agency for Infrastructure Technology Advancement (KAIA) (Grant No. 20CTAP-C152021-02), the Basic Science Research Program through the National Research Foundation of Korea (Grant No. 2020R1A6A1A03046811), and the Institute of Civil Military Technology Cooperation (Development of Highly Reliable UAV Flight Simulator Technology with Wearable Devices).

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: The datasets generated and/or analysed during the current study are available from the corresponding author on reasonable request.

Acknowledgments: All of the authors would like to appreciate and thank the anonymous reviewers for giving such detailed and constructive comments and suggestions throughout the peer-review processes.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Park, J.; Kim, J.; Song, J. Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning. *Int. J. Control Autom. Syst.* **2007**, *5*, 674–680.
2. Kato, Y.; Kamiyama, K.; Morioka, K. Autonomous robot navigation system with learning based on deep Q-network and topological maps. In Proceedings of the 2017 IEEE/SICE International Symposium on System Integration (SII), Taipei, Taiwan, 11–14 December 2017; pp. 1040–1046. [CrossRef]
3. Bello, I.; Pham, H.; Le, Q.V.; Norouzi, M.; Bengio, S. Neural Combinatorial Optimization with Reinforcement Learning. *arXiv* **2016**, arXiv:1611.09940.
4. Huang, H.; Yang, Y.; Wang, H.; Ding, Z.; Sari, H.; Adachi, F. Deep Reinforcement Learning for UAV Navigation Through Massive MIMO Technique. *IEEE Trans. Veh. Technol.* **2020**, *69*, 1117–1121. [CrossRef]
5. Faust, A.; Palunko, I.; Cruz, P.; Fierro, R.; Tapia, L. Automated aerial suspended cargo delivery through reinforcement learning. *Artif. Intell.* **2017**, *247*, 381–398. [CrossRef]
6. Faust, A.; Ramirez, O.; Fiser, M.; Oslund, K.; Francis, A.; Davidson, J.; Tapia, L. PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018.
7. Kavraki, L.E.; Svestka, P.; Latombe, J.; Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **1996**, *12*, 566–580. [CrossRef]
8. Dai, H.; Khalil, E.B.; Zhang, Y.; Dilkina, B.; Song, L. Learning Combinatorial Optimization Algorithms over Graphs. *arXiv* **2017**, arXiv:1704.01665.
9. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer Networks. *arXiv* **2015**, arXiv:1506.03134.
10. Francis, A.; Faust, A.; Chiang, H.T.L.; Hsu, J.; Kew, J.C.; Fiser, M.; Lee, T.W.E. Long-Range Indoor Navigation with PRM-RL. *IEEE Trans. Robot.* **2020**, *36*, 1115–1134. [CrossRef]

11. Chiang, H.T.L.; Hsu, J.; Fiser, M.; Tapia, L.; Faust, A. RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators from RL Policies. *IEEE Robot. Autom. Lett.* **2019**, *4*, 4298–4305. [CrossRef]
12. Maw, A.A.; Tyan, M.; Lee, J.W. iADA*: Improved Anytime Path Planning and Replanning Algorithm for Autonomous Vehicle. *J. Intell. Robot. Syst.* **2020**, *100*, 1005–1013. [CrossRef]
13. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*; Springer: Cham, Switzerland, 2017.
14. Sanders, A. *An Introduction to Unreal Engine 4*; A. K. Peters, Ltd.: Natick, MA, USA, 2016.
15. Juliani, A.; Berges, V.; Vckay, E.; Gao, Y.; Henry, H.; Mattar, M.; Lange, D. Unity: A General Platform for Intelligent Agents. *arXiv* **2018**, arXiv:1809.02627.
16. Fielding, R.T. Architectural Styles and the Design of Network-Based Software Architectures. Ph.D. Thesis, University of California, Irvine, MA, USA, 2000.
17. Meier, L.; Tanskanen, P.; Fraundorfer, F.; Pollefeys, M. PIXHAWK: A system for autonomous flight using onboard computer vision. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 2992–2997. [CrossRef]
18. Qin, T.; Li, P.; Shen, S. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Trans. Robot.* **2018**, *34*, 1004–1020. [CrossRef]
19. García, J.; Molina, J.M. Simulation in real conditions of navigation and obstacle avoidance with PX4/Gazebo platform. *Pers. Ubiquitous Comput.* **2020**. [CrossRef]
20. Mendoza-Mendoza, J.A.; Gonzalez-Villela, V.; Sepulveda-Cervantes, G.; Mendez-Martinez, M.; Sossa-Azuela, H. ArduPilot Working Environment. In *Advanced Robotic Vehicles Programming: An Ardupilot and Pixhawk Approach*; Apress: Berkeley, CA, USA, 2020; pp. 19–46. [CrossRef]
21. Bennett, S. Development of the PID controller. *IEEE Control Syst. Mag.* **1993**, *13*, 58–62. [CrossRef]
22. Rohmer, E.; Singh, S.P.N.; Freese, M. V-REP: A versatile and scalable robot simulation framework. In Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, 3–7 November 2013; pp. 1321–1326. [CrossRef]
23. Szepesvári, C. Algorithms for Reinforcement Learning. *Synth. Lect. Artif. Intell. Mach.* **2010**, *4*, 1–103. [CrossRef]
24. Kober, J.; Bagnell, J.A.; Peters, J. Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
25. Kormushev, P.; Calinon, S.; Caldwell, D. Reinforcement Learning in Robotics: Applications and Real-World Challenges. *Robotics* **2013**, *2*, 122–148. [CrossRef]
26. Sanghi, N. Markov Decision Processes. In *Deep Reinforcement Learning with Python: With PyTorch, TensorFlow and OpenAI Gym*; Apress: Berkeley, CA, USA, 2021; pp. 19–48. [CrossRef]
27. Koenig, N.; Howard, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Coimbra, Portugal, 8–12 September 2004; Volume 3, pp. 2149–2154. [CrossRef]
28. Hernandez, A.; Copot, C.; De Keyser, R.; Vlas, T.; Nascu, I. Identification and path following control of an AR.Drone quadrotor. In Proceedings of the 2013 17th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 11–13 October 2013; pp. 583–588. [CrossRef]
29. Qiu, W.; Yuille, A. UnrealCV: Connecting Computer Vision to Unreal Engine. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016.
30. Krishnan, S.; Boroujerdian, B.; Fu, W.; Faust, A.; Reddi, V.J. Air Learning: An AI Research Platform for Algorithm-Hardware Benchmarking of Autonomous Aerial Robots. *arXiv* **2019**, arXiv:1906.00421
31. Boroujerdian, B.; Genc, H.; Krishnan, S.; Cui, W.; Faust, A.; Reddi, V.J. MAVBench: Micro Aerial Vehicle Benchmarking. *arXiv* **2019**, arXiv:1905.06388.
32. Hornung, A.; Wurm, K.M.; Bennewitz, M.; Stachniss, C.; Burgard, W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Auton. Robot.* **2013**, *34*, 189–206. [CrossRef]
33. Fujimura, K.; Kunii, T.; Yamaguchi, K.; Toriya, H. Octree-Related Data Structures and Algorithms. *IEEE Comput. Graph. Appl.* **1984**, *4*, 53–59. [CrossRef]
34. Chen, T.; Shan, J. A novel cable-suspended quadrotor transportation system: From theory to experiment. *Aerosp. Sci. Technol.* **2020**, *104*, 105974. [CrossRef]