# An Efficient Deep Reinforcement Learning Framework for UAVs

Shanglin Zhou[1], Bingbing Li[1], Caiwu Ding[2], Lu Lu[2], Caiwen Ding[1]
[1]University of Connecticut [2]New Jersey Institute of Technology
E-mail: [1]{shanglin.zhou, bingbing.li, caiwen.ding}@uconn.edu
[2]caiwu.ding@njit.edu [2]lulu.lvlv@gmail.com

**Abstract**— 3D Dynamic simulator such as Gazebo has become a popular substitution for unmanned aerial vehicle (UAV) because of its user-friendly in real-world scenarios. At this point, well-functioning algorithms on the UAV controller are needed for guidance, navigation, and control for autonomous navigation. Deep reinforcement learning (DRL) comes into sight as its famous self-learning characteristic. This goal-orientated algorithm can learn how to attain a complex objective or maximize along a particular dimension over many steps. In this paper, we propose a general framework to incorporate DRL with the UAV simulation environment. The whole system consists of the DRL algorithm for attitude control, packing algorithm on the Robot Operation System (ROS) to connect DRL with PX4 controller, and a Gazebo simulator that emulates the real-world environment. Experimental results demonstrate the effectiveness of the proposed framework.

**Keywords**— Deep Reinforcement Learning, Gazebo, PX4, Simulation Environment, Unmanned Aerial Vehicle

## I. Introduction

Experiments on physical UAVs such as UAVs are usually time-consuming and expensive because the parameters need to be accommodated to different environmental settings [1]. The trial and error process lead to a high capital cost, makes it prohibitive for those machines that are expensive and rare. In order to reduce the capital cost, different simulation environments are widely used to analyze and validate the performance of the UAV before the physical machine is built. A well-designed simulator provides realistic scenarios and makes it possible to rapidly test algorithms, design UAVs, and train AI system.

Gazebo is a 3D multi-UAV simulator with dynamic build-in functions, capable of simulating articulated UAV in complex and realistic environments [1]. Using multiple high-performance physics engines such as Open Dynamics Engine (ODE) and Bullet, Gazebo can be used to simulate super realistic environment including high-quality lighting, shadows, and textures. Various sensor functions are integrated in the simulator, such as laser range finders, cameras (including wide-angle), Kinect style sensors, etc [2].

In addition, a control strategy orchestrates the underneath controller for UAV control, where proportional-integral-derivative (PID) controllers are usually used to achieve desired thrust and three Euler angles. Recently, machine learning methods have been borrowed as the UAV control strategy. Among which, reinforcement learning (RL) becomes popular where an agent learns how to behave in an environment by performing actions and receiving rewards [3]. As shown in Figure 1, environment presents current state $s_t$ and reward $r_t$ to agent, agent then take action $a_t$ to maximize a discounted accumulative reward. Then based on $a_t$, environment outputs state $s_{t+1}$ and reward $r_{t+1}$ for next time stamp. Q-Learning is a value-based RL, where "Q" stands for the "quality" of an action taken in a given state, which is the function that returns the reward used to provide the reinforcement.

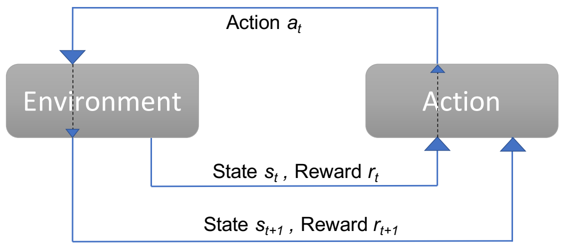
Action $a_t$
Environment
Action
State $s_t$ , Reward $r_t$
State $s_{t+1}$ , Reward $r_{t+1}$

Fig. 1. Agent-Environment Interaction System. Environment returns state and reward, agent selects action to maximize a discounted accumulative reward.

Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action given the current state. Through Q-learning, the controller can directly response with action without hand-crafted engineering work, to enable UAVs to self-learn the desired rotor Euler angles/action. Q-learning aims to find an optimal policy that it maximizes the expected value of the total reward over all successive steps, starting from the current state[4]. However, when the state and action space are high-dimensional and continuous, it is not realistic to use Q-Table. Deep neural networks (DNNs) are excellent for the ability to extract hidden features. Combination of DNN and Q-learning, a.k.a., deep reinforcement learning (DRL) or deep Q-Network (DQN), enables the network to directly learning control strategies from high-dimensional raw data [5]. It takes raw input as states and output the value evaluation (Q-value) corresponding to each action.

In this paper, we adopt Gazebo, an open-source 3D dynamic simulator to calculate the response of the UAV, and we use PX4 firmware, a programmable autopilot system with packs of PID controllers to control the Euler angles and thrust of the UAV. In addition, we propose a general framework to incorporate DQN algorithm with the UAV simulation environment. The DQN is utilized to receive the state of the UAV and give high-level commands (thrust) to the PX4 firmware. The whole framework is shown in Figure 2. The rest of the paper is organized as following: In Section II, development of DRL are briefly introduced, including the development of DQN algorithm and how it gained popularity. Background of UAV controller environment as well as DQN algorithm are introduced in details in Section III. Then a comprehensive system model is conducted in Section IV including internal connection and DRL node. The experiments are presented in Section V and finally Section VI concludes this work.

## II. Related Work

As early as 1989, Watkins proposed algorithm of Q-learning [6] including model-free reinforcement learning, asynchronous dynamic programming method and Markovian domains based action-environment interaction learning. Later in 1992, Watkins *et al.* [7] provided the detailed convergence proof of the Q-learning algorithm, which proves that when all states can be repeatedly accessed, the Q function will eventually converge to the optimal Q-value. The best-known successful application of Q-learning is TD-gammon [8]. This backgammon-playing program achieved a super-human level of play by using multi-layer perceptron with one hidden layer to approximate state value function, and a model-free reinforcement learning algorithm very similar to Q-learning.

Restricted Boltzmann machines was also used in estimating the value function [9] or policy [10] in Q-learning. Since 2012, deep learning has made great progress and received unprecedented attention [11]. Earlier similar work as deep Q-learning is the neural fitted Q-learning (NFQ) [12]. Due to the use of RPROP algorithm in updating the Q-network's parameters, computation cost per iteration is proportional to the size of the data set, which is quite time-consuming when data set is large. An extension work of NFQ is its combination with deep auto-encoder [13].

Although combining deep learning and reinforcement learning together has been applied to real world control tasks for many years, the beginning of real success is actually DeepMinds' Atari playing using Deep Q-learning (DQN) [14]. It used convolutional neural networks (CNNs) to extract effective features directly from high-dimensional sensory inputs, then band with Q-Learning to obtain the optimal strategy of policy. In early 2015, DeepMind developed an improved version of DQN [15]. With bringing in the *experience replay* and *target network*, stability of the original DQN in actual training is improved. To further enhance convergence and instability of DQN, Hasselt *et al.* [16] proposed to repeatedly freeze the target network, Schaul *et al.* [17] proposed a prioritized experience replay method to improve sampling efficiency, and Wang *et al.* [18] proposed Dueling DQN architecture to learn more robust state value function values by decomposing Q function into state value function and state-dependent action advantage function. In 2016, Mnih *et al.* [19] proposed a asynchronous advantage actor-critic deep reinforcement learning method (A3C) that achieved super impressive results in most of games

A prior work similar to our approach is [20]. It wrapped the deep deterministic policy gradients (DDPG) method with Gazebo simulator. Comparing with that, we introduce deep Q-learning, which is easier to implement actions responded by environment since DDPG needs the action space to be continuous, and also DQN is faster in converge. Another similar work is [21], where the authors directly use build-in controller functions. The PX4 controller makes our framework much more flexibility in thrust and angle control, which enables more states and actions to be applied when building DRL algorithm.

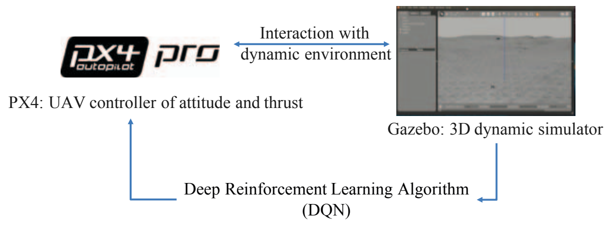
PX4 autopilot pro
Interaction with dynamic environment
PX4: UAV controller of attitude and thrust
Gazebo: 3D dynamic simulator
Deep Reinforcement Learning Algorithm (DQN)

Fig. 2. Architecture of proposed framework. Gazebo provides 3D simulation environment interact with PX4 controller, DRL algorithm acts as a bridge to provide guidance for the interaction.

## III. UAV Control Environment and DRL Algorithm

### A. UAV Control Environment

Gazebo provides the three most basic graphic components: spheres, cylinders, and cubes. Using these three graphic components and their telescoping or rotation transformations, a simple three-dimensional simulation model of the UAV can be designed by users. Realistic rendering of environments could be created thanks to the provided lighting, shadows and textures. Additionally, Gazebo offers plenty of 2D and 3D software interfaces such as CAD, Blender, so that various designing software and coding editors can be connected to make modifications on the UAV or algorithms much more convenient [1].

Gazebo can build a simulation scenario for testing UAVs, imitate the real world by adding object libraries, gravity and resistance. Extensive command line tools are also available to make it easier to facilitate simulation introspection and control. 3D models can be constructed based on the 2D design drawings through its build-in "Building Editor". Besides, multiple high-performance physics engines including ODE, Bullet, Simbody, and DART can be accessed. Moreover, Gazebo has a very powerful sensor model library, including some sensors commonly used by UAVs such as camera, depth camera, laser, imu, etc. And there actually exists an integrated simulation library that can be used directly, or we can also create a new sensor from null, add specific parameters, evenly add sensor noise models to make the sensor more realistic [1].

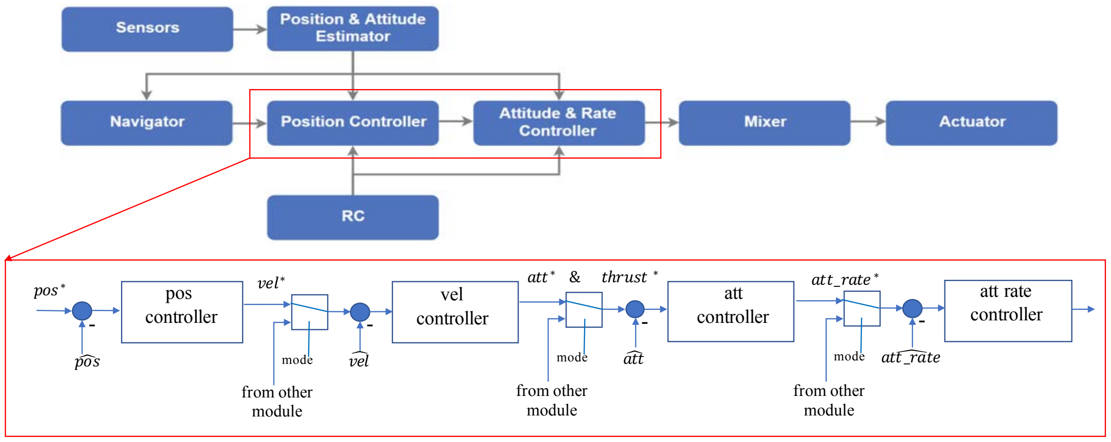

Fig. 3. PX4 Controller Architecture. Generally been divided into several parts.

PX4 is then adopted to interact with dynamic simulation environment. PX4 is a Professional Autopilot [22] that provides a set of tools for UAV developers to create their own drone applications. PX4 Firmware provides plenty of functions to manage different processes, to build basic position, velocity, attitude (rotation angle), and attitude rate (angle acceleration) controllers, to establish the communication between different processes through a publish-subscribe message bus named uORB, and to create the interface with external devices.

PX4 Firmware consists of four cascade PID controllers, respectively for controlling UAV's position, velocity, attitude and attitude rate. Figure 3 displays the traditional process of using PX4 Firmware to control UAV in the simulation environment. The thrust controller receives thrust command and generates command of rotation speed to the Electronic Speed Controller (ESC) of the brushless motors. Similarly, other control loops are utilized, including a position (pos) controller receiving outside position command and generating velocity (vel) command. The velocity controller receives velocity command and generates attitude and thrust commands. The attitude and thrust commands are processed by some other modules, and transmitted to the attitude and thrust controllers. for attitude rate command generation.

Due to time consuming, difficult in operation and slowly in convergence, we propose DRL algorithm to replace position and velocity controllers in PX4 Firmware. With attitude and attitude rate controllers retained, the modified PX4 Firmware directly receives thrust from DRL instead of from the position and velocity controllers.

Moreover, in order to connect Gazebo and PX4 with the DRL controlling algorithm, Robot Operating System (ROS) is adopted. ROS is an open-source, meta-operating system, which supports hardware abstraction, low-level device control, message-passing between processes and management. ROS is a distributed framework of processes (Nodes) that enables executable to be individually designed and loosely coupled at running time [23].

## B. DRL Algorithm for UAV Control

In DQN, we adopt convolutional neural network (CNN) to perform function approximation of Q-Table in high-dimensional and continuous space, while Q-learning is used to determine the loss function and update of the parameters. Q-table is used to store the Q-value of each action-state pair when the spaces of both action and state are discrete and low dimensional. As a reference table, the agent can select the best action based on Q-value in the table. One can input the state of each time stamp into a CNN to train and estimate the rewards according to position changes as Q-value. CNN is a supervised learning method that needs a bunch of labeled data to train the network [11]. DQN uses target Q-value as labels. The loss function of the whole process, if mean square error loss is used, can be written as

$$L(\theta) = [Q_{target} - Q(s_t, a; w_{eval})]^2, \quad t = 1, ..., T \tag{1}$$

where $Q_{target} = r + \gamma \max_{a'} Q(s_{t+1}, a'; w_{target})$ is from the target Q net. $Q(s_t, a; w_{eval})$ is the evaluated Q-value of current time stamp. As same notation in Figure 1, $s_t$ and $s_{t+1}$ are the states (position, velocity, rotation angles, angle rate, angle acceleration) of the current and next time stamp; $a$ and $a'$ are the UAV actions (decrease thrust, increase thrust, or restart) of time stamp $t$ and $t+1$; $w_{eval}$ and $w_{target}$ are the weights of the evaluation Q net and target Q net, respectively. $\gamma$ is discount factor. $r$ is reward given state and action. These two nets will be described in detail in Section 3-B.3.

### B.1 Experience Replay

Experience replay is introduced in DQN for correlation of past experience and non-static distribution of Q-learning problems. At each time stamp, action interacts with environment, then a transfer tuple *(state, action, reward, new_state)* can be received and stored to a "replay buffer". During training process, a mini-batch of transfer tuples is sampled randomly from the buffer and then fed to the neural network. The whole "interaction between environment and agent" process is divided into fragments and stored separately, to prevent action values from oscillating or diverging catastrophically and make more efficient use of observed experiences.

### B.2 $\epsilon$-Greedy Strategy

A policy is required to select actions. There are two approaches in selecting the optimal action. The first one is random selection, the second one is calculating an optimal action based on the current Q-value, called greedy strategy:

$$\pi(s_{t+1}) = arg \max_a Q(s_{t+1}, a; w) \tag{2}$$

In $\epsilon$-greedy strategy, we select a random action with a very small probability $\epsilon$ and choose the current optimal estimate action with probability $1-\epsilon$. The random selection is called "exploration", to explore the effect of unknown actions, update the Q-value, and obtain a more robust policy. The greedy strategy is named "exploitation", leading to more accurate test results.

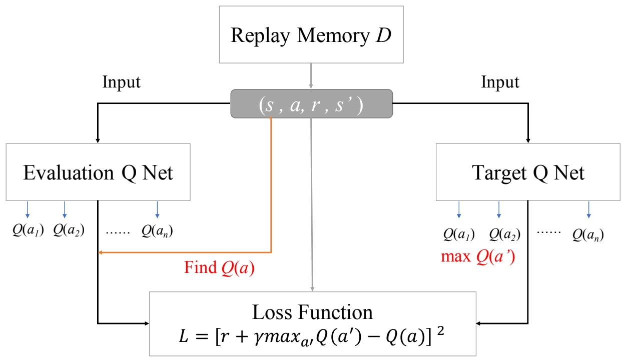
Replay Memory $D$
Input
Input
$(s, a, r, s')$
Evaluation Q Net
Target Q Net
$Q(a_1)$ $Q(a_2)$ ...... $Q(a_n)$
$Q(a_1)$ $Q(a_2)$ ...... $Q(a_n)$
Find $Q(a)$
max $Q(a')$
Loss Function
$L = [r + \gamma max_{a'} Q(a') - Q(a)]^2$

Fig. 4. Loss Function update for the two net. Evaluation Q net outputs evaluated Q-value, target Q net outputs max Q-value.

### B.3 Dual Network Structure

There are two CNNs in the entire process. One is the evaluation Q net, which learns from past experiences, feeds current state as input, finds hidden distribution of Q-table and outputs respective evaluated Q-value for each possible action. The other one is the target Q net, which is built in providing labels (target Q-value) for evaluation Q net. Target Q net has exactly the same structure as the evaluation Q net but with delays in weight updating, i.e., the parameters of the evaluation Q net are copied to the target Q net after several training epochs. The target Q-value remains unchanged for several epochs, which reduces the correlation between the evaluated Q-value and the target Q-value to a certain extent and improves the stability. It is worth noticing that input of the evaluation Q net is the current state while it is next state for target Q net. The loss function update for the two nets are shown in Figure 4.

The full process of deep Q-learning is presented in Algorithm 1. Q-Learning algorithm is called the Off-policy algorithm. Because Q-Learning only considers the current environment and reward instead of the entire environment. Thus, DQN is a model-free method [15].

**Algorithm 1** Deep Q-learning Algorithm for UAV Control

```
Initialize replay memory $D$ to capacity $N$
Initialize $w_{target}$ and $w_{eval}$, let $w_{target} = w_{eval}$
for $episode = 1, M$ do
    Initialize state (position, velocity, rotation angles, angle rate, angle acceleration) $s_1$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
          from {decrease thrust, increase thrust, restart}
        Otherwise select $a_t = arg \max_a Q(s_t, a, w_{eval})$
        Execute action $a_t$ to simulator and observe reward
          $r_t$ from position changes, and state $s_{t+1}$
        Store transfer tuple $(s_t, a_t, r_t, s_{t+1})$ in $D$
        Sample random minibatch of transfer tuples
          $(s_j, a_j, r_j, s_{j+1})$ from $D$
        Set
          $Q_{target} = \begin{cases} r_j & if\ episode\ terminates\ at\ step\ j+1 \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; w_{target}) & o.w \end{cases}$
        Perform $SGD$ step on the loss function as Eqn 1
          with respect to the network parameters $w_{eval}$
        Every $C$ steps reset $w_{target} = w_{eval}$
    end for
end for
```

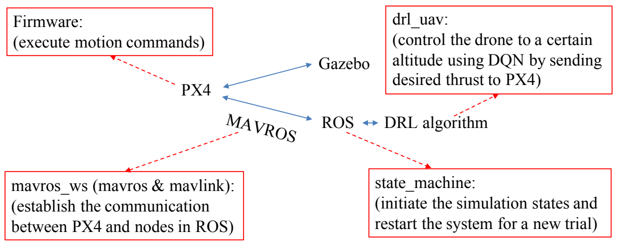
Firmware:
(execute motion commands)
drl_uav:
(control the drone to a certain altitude using DQN by sending desired thrust to PX4)
Gazebo
PX4
MAVROS
ROS
DRL algorithm
mavros_ws (mavros & mavlink):
(establish the communication between PX4 and nodes in ROS)
state_machine:
(initiate the simulation states and restart the system for a new trial)

Fig. 5. Framework Internal Connection. Gazebo performs simulation environment, interact with PX4, a controller that executes motion commands. ROS provides operating system to pack DRL algorithm as node then transfers messages with PX4 using MAVROS link.

## IV. **System Model**

### A. *Internal Connection*

The used modules and the inter-connections are shown in Figure 5. The raw data of the UAV model are obtained in Gazebo environment using sensors. A state estimator using Extended Kalman Filter (EKF) is utilized to estimate the state of the UAV (position, velocity, rotation angles, angle rate, angle acceleration, etc). A cascade PID controller is used to control the velocity and thrust after receiving commands from the DRL algorithm. To establish the communication between the PID controller and DRL, we convert our DRL algorithm into a node in ROS and use MAVROS to transfer messages between PX4 firmware and the DRL node.

## B. DRL Node

Figure 6 is a subset part of Figure 3. The state estimator, attitude controller and attitude rate controller are retained, while the position and velocity controllers are replaced by the DRL algorithm. This makes the position controller and attitude rate controller more stable in practical application, and more compatible with the Gazebo dynamic simulator. We directly use raw Gazebo output as the input of our DQN algorithm, i.e., $\{$*position, velocity, rotation angles, angle rate, angle acceleration*$\}$ at the current time stamp.

Each CNN layer is followed by rectified nonlinearity (ReLU). The output layer is a fully connected linear layer with a single output for each valid action. Valid actions are denoted as $\{0, 1, -1\}$, where 0 represents *decrease thrust*, 1 represents *increase thrust* and $-1$ represents *environment needs to be restarted*. We set the restart action when *altitude* is larger than 30 or smaller than 10 (far away from our intended hovering location, i.e., $20m$.) In training process, *RMSProp* optimization method, an adaptive learning rate method, is used with mini-batches of size 32 to enable faster training and convergence.

In Q-learning, $\epsilon$-greedy policy is introduced in action selection in order to reduce random action in greedy algorithm. Here, we exponentially anneal $\epsilon$ from 1 to 0.001 following the function

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min})\, e^{-\lambda * t} \tag{3}$$

where $\epsilon_{min}$ is the minimum value of $\epsilon$. $\epsilon_{max}$ is the maximum value of $\epsilon$. $\lambda$ is decay speed which is set to 0.0015, and $t$ represents time stamp.

Since UAV will hovering around altitude of 20, 1.0 reward is earned when value of altitude of UAV is between 19.7 and 20.3, otherwise 0.0 reward earned. Besides, a first-in-first-out replay buffer with capacity of one million is used to store the most recent response from the simulator *i.e., state, action, reward, new_state.*

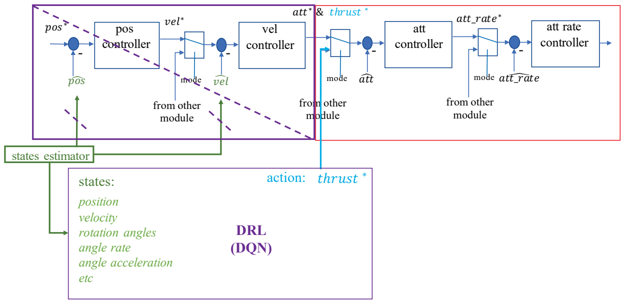

Fig. 6. DRL Attitude and Attitude Rate Control Loop. Use DQN for attitude control. State estimator, attitude controller and attitude rate controller are retained, position and velocity controllers are replaced by DRL algorithm

## V. **Evaluation**

The overview of the whole framework is shown in Figure 5. To guarantee the functioning the whole simulation

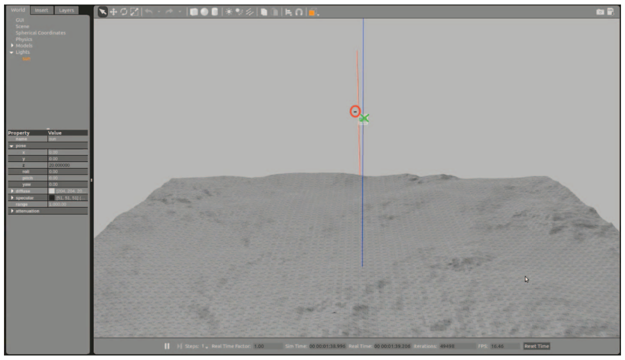

Fig. 7. Experimental environment from the Gazebo Simulator. Red circle indicates location of drone, while green cross is the hovering destination, which has coordinate $(0, 0, 20)$.

environment, more attention should be paid to ensure the compatibility. In the experiment, we use ROS Kinetic, which is a long-term support release and compatible with Ubuntu 16.04. To obtain real-time states of the UAV, different sensors are built on the simulation, including gyroscope, accelerometer, magnetometer (compass), barometer and GPS, while more sensors, e.g. LIDAR, RGB-D and stereo cameras, could also be added to the UAV for other applications. The original controller consists of four loops as described in Figure 3. Each controller loop receives the expected value from the outer loop and also g an expected value as the command to the inner loop. To make full use of the basic functions of the attitude and attitude rate controllers, thrust commands are directly given by the DRL algorithm to the PX4 firmware. In this way, the DRL algorithm avoids to give rotation commands to the brushless motors directly and could be treated as a high-level decision maker. MAVROS is used to establish the communication between DRL algorithm and PX4 firmware to transfer UAV states information and commands. To take advantage of the ROS architecture and communications infrastructure, we use different DRL nodes to create several processes including DRL algorithm and MAVROS. All these environments are tested in a host machine with Core i9-8950HK CPU and RTX 2080 Max-Q GPU.

We use the DRL algorithm to control the UAV to achieve hovering at a position P (0m, 0m, 20m). The attitude and the horizontal position are controlled using PID controller in PX4 Firmware. The vertical position is controlled using DRL by giving thrust commands to the PX4 Firmware. The initial position of the UAV is chosen at point P and then the action of the thrust and the states are obtained while training. The DRL algorithm selects different actions, estimate the reward of each action based on the current state and choose the optimal one.

To prevent the divergence of the state estimator, we choose to increase the thrust (action $= 1$) when the velocity achieves a very large negative number ($-3.0m/s$), and to decrease (action $= 0$) when the velocity achieves a very large positive number ($3.0m/s$). Also, to ensure the

TABLE I

Examples of collected states

| Index | Height(m) | Vertical Velocity(m/s) | Thrust Setpoint | Action | Reward |
|---|---|---|---|---|---|
| 1 | 20 | 0.1 | 0.5 | 1 | 1 |
| 2 | 21 | 0.5 | 0.8 | 1 | 0 |
| 3 | 19 | -0.3 | 0.3 | 1 | 0 |
| ... | ... | ... | ... | ... | ... |

TABLE II

Total reward (TR) with respect to training epochs

| Epoch | Highest TR |
|---|---|
| 1 | 7 |
| 500 | 375 |
| 1000 | 1154 |
| 2000 | 2277 |
| 3000 | 4314 |

convergence of the DRL algorithm, we stop training when the current height of the UAV is out of the safe range ($[10m, 30m]$), utilize PID controller to force the UAV to the original position ($[0m, 0m, 20m]$), and restart the DRL algorithm. The simulation results are shown in Figure 7.

The states of the UAV, i.e. position, velocity, rotation angles, angle rate, angle acceleration, are collected as the input of the DRL algorithm. Table I shows several examples of the collected states. To evaluate the effectiveness of the proposed method, we use Total Reward (TR), which is the sum of the rewards during certain training epochs. With the increase of the training epochs, the DRL algorithm is able to achieve higher TR to make better state-action decisions as shown in Table II.

## VI. Conclusion

In this paper, we propose a UAV simulation framework that combines deep reinforcement learning algorithm with Gazebo simulator. Gazebo provides a realistic 3D dynamic simulation environment, and interacts with PX4 firmware to give rotor rotation commands to the UAV as a low-level controller. We replace position and velocity controllers in PX4 with DRL algorithm, use position, velocity, rotation angles, angle rate, angle acceleration from simulator as input and directly output thrust for attitude controller. We also use the ROS to connect DRL with PX4 controller. Experiments show that the framework works effectively.

## References

[1] "Gazebo: Robot simulation," http://gazebosim.org/.

[2] Evan Ackerman, "Latest version of gazebo simulator makes it easier than ever to not build a robot," 2016, https://spectrum.ieee.org/automaton/robotics/robotics-software/latest-version-of-gazebo-simulator.

[3] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[4] Francisco S Melo, "Convergence of q-learning: A simple proof," *Institute Of Systems and Robotics, Tech. Rep*, pp. 1–4, 2001.

[5] David Silver, "Deep reinforcement learning," *Tutorial at ICML*, 2016.

[6] CJCH Watkins, "Learning from delayed rewards (phd dissertation)," *King's College Cambridge, England*, 1989.

[7] Christopher JCH Watkins and Peter Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[8] Gerald Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[9] Brian Sallans and Geoffrey E Hinton, "Reinforcement learning with factored states and actions," *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 1063–1088, 2004.

[10] Nicolas Heess, David Silver, and Yee Whye Teh, "Actor-critic reinforcement learning with energy-based policies.," in *EWRL*, 2012, pp. 43–58.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[12] Martin Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning.* Springer, 2005, pp. 317–328.

[13] Sascha Lange and Martin Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *The 2010 International Joint Conference on Neural Networks (IJCNN).* IEEE, 2010, pp. 1–8.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller, "Playing atari with deep reinforcement learning," *ArXiv*, vol. abs/1312.5602, 2013.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529, 2015.

[16] Hado Van Hasselt, Arthur Guez, and David Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[17] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[18] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[19] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[20] DanielLSM, "Deep reinforcement learning scripts working with ros+gazebo+"gym like wrappers"," 2017, https://github.com/DanielLSM/Gazebo-DRL.

[21] Risto Kojcev, Nora Etxezarreta, Alejandro Hernández, and Víctor Mayoral, "Evaluation of deep reinforcement learning methods for modular robots," *arXiv preprint arXiv:1802.02395*, 2018.

[22] Dronecode, "Px4 autopilot user guide (master)," http://docs.px4.io/master/en/index.html.

[23] ROS.org, "Robot operating system (ros) introduction documentation," http://wiki.ros.org/ROS/Introduction.