drones

*Article*

# Drone Elevation Control Based on Python-Unity Integrated Framework for Reinforcement Learning Applications

**Mahmoud Abdelkader Bashery Abbass [1,2] and Hyun-Soo Kang [1,*]**

[1] Department of Information and Communication Engineering, School of Electrical and Computer Engineering, Chungbuk National University, Cheongju-si 28644, Republic of Korea; mahmoud.gohar1992@m-eng.helwan.edu.eg
[2] Department of Mechanical Power Engineering, Helwan University, Cairo 11772, Egypt
* Correspondence: hskang@cbnu.ac.kr; Tel.: +82-43-261-3488

**Abstract:** Reinforcement learning (RL) applications require a huge effort to become established in real-world environments, due to the injury and break down risks during interactions between the RL agent and the environment, in the online training process. In addition, the RL platform tools (e.g., Python OpenAI's Gym, Unity ML-Agents, PyBullet, DART, MoJoCo, RaiSim, Isaac, and AirSim), that are required to reduce the real-world challenges, suffer from drawbacks (e.g., the limited number of examples and applications, and difficulties in implementation of the RL algorithms, due to difficulties with the programing language). This paper presents an integrated RL framework, based on Python–Unity interaction, to demonstrate the ability to create a new RL platform tool, based on making a stable user datagram protocol (UDP) communication between the RL agent algorithm (developed using the Python programing language as a server), and the simulation environment (created using the Unity simulation software as a client). This Python–Unity integration process, increases the advantage of the overall RL platform (i.e., flexibility, scalability, and robustness), with the ability to create different environment specifications. The challenge of RL algorithms' implementation and development is also achieved. The proposed framework is validated by applying two popular deep RL algorithms (i.e., Vanilla Policy Gradient (VPG) and Actor-Critic (A2C)), on an elevation control challenge for a quadcopter drone. The validation results for these experimental tests, prove the innovation of the proposed framework, to be used in RL applications, because both implemented algorithms achieve high stability, by achieving convergence to the required performance through the semi-online training process.

**Keywords:** reinforcement learning agent; simulation platform; Python programming language; Unity simulation software; UDP communication protocol; Vanilla Policy Gradient algorithm; Actor–Critic algorithm



**Citation:** Abbass, M.A.B.; Kang, H.-S. Drone Elevation Control Based on Python-Unity Integrated Framework for Reinforcement Learning Applications. *Drones* **2023**, *7*, 225. https://doi.org/10.3390/drones7040225

Academic Editor: Andrey V. Savkin

Received: 30 January 2023
Revised: 11 March 2023
Accepted: 15 March 2023
Published: 24 March 2023



## 1. Introduction

In recent years, significant research in the reinforcement learning (RL) domain, has had a significant impact on different applications [1] (e.g., healthcare [2–6], gaming [7–9], natural language processing [10–13], self-driving cars [14–17], and robotics [18–23]). At the same time, applying RL requires high safety considerations during the training and implementation steps, especially in applications that require online training steps or interactions between the agent and the real-world environment (e.g., self-driving cars and robotics). These safety considerations come from the risk of insufficient action from the agent, with respect to the instant state, before achieving convergence to the best performance. Therefore, the concept of performing the RL training process in a real-world (i.e., online training) environment is challenging, because of the RL model adjustment required, with respect to a continuous feed for real-time data and continuous change of the environment's parameters. For example, the incorporation of an RL algorithm in an unmanned aerial vehicle (UAV)

control, in real-time (online training), has a high-risk probability (e.g., breakdown or human injury risks). So, most of the real-life implementations of RL algorithms in UAVs, include many safety considerations [23,24].

Meanwhile, research has also focused on the RL simulation platforms (e.g., Python OpenAL Gym, and Unity ML-Agents), as a preprocessing step for the agent's behavior in a real-world environment. However, the existing technology in the RL simulation platforms either suffers from limited environment abilities and examples (e.g., Python OpenAI Gym, PyBullet, DART, MoJoCo, RaiSim, Isaac, and AirSim), or is based on difficult programming languages (e.g., Unity ML-Agents). Hence, the proposed framework must overcome all of the drawbacks of the existing platforms. So, a review of the current state of the art is given in the rest of the introduction section, to clearly explain the contribution and innovation of the proposed framework, with a comparison to the existing technology.

### *1.1. Related Work*

To create an efficient framework for RL applications, the most utilized tools in this area must be demonstrated. Hence, the most popular platforms (Table 1) and communication protocols are described in the following sub-sections.

#### 1.1.1. Existing Platforms

OpenAI Gym: This is a free Python software package from OpenAI, that uses several reinforcement learning benchmarks, including traditional control, Atari, robotics, and MuJoCo tasks. Environments for creating and testing learning agents are present in OpenAI. Although it is concentrated, and most appropriate for reinforcement learning agents, it does not prevent one from experimenting with alternative techniques, such as hard-coded game solvers or other deep learning techniques [25,26]. The environments are limited (e.g., cartpole, mountain car, pendulum, acrobat, car racing, pi-pedal walker), and adding new environments is hard. So, most research work uses this platform for implementing and developing the RL agent algorithms [27–32].

Unity ML-Agents: An open-source project called Unity Machine Learning Agents Toolkit (ML-Agents), which makes it possible to use simulations and games as training grounds for intelligent agents. The behavior of NPCs may be controlled (in many contexts, such as multi-agent and adversarial), the game builds can be automatically tested, and various game design choices can be assessed, before release, using these trained agents. The ML-Agents Toolkit offers a single platform, where developments in AI can be assessed on Unity's rich settings and then made available to the larger research and game developer communities, which benefits both game creators and AI researchers [33]. Due to the environment creation flexibility in Unity, this tool has been used in different applications, such as a tennis competition and soccer competition [34], a billiards competition, transport bricks, a capture flag competition [35], an obstacle tower environment [36], and a marathon race [37]. At the same time, it contains a limited number of RL algorithms (e.g., Proximal Policy Optimization (PPO) and Soft Actor–Critic (SAC)), and adding more algorithms is hard because of the challenging programming language (i.e., C#). So, it is not suitable for the implementation and development of RL algorithms [33].

PyBullet: This is a type of platform that was created based on an integration with the Gazebo platform [38], to perform a full simulation. Based on the PyBullet Physics SDK, PyBullet is a simple-to-use Python package for robotics, deep reinforcement learning, and physics simulation. The PyBullet robotic examples, such as a simulated Minitaur quadruped, humanoids running using TensorFlow inference, and KUKA arms gripping items, are also part of the PyBullet Physics SDK. Users may import articulated bodies from URDF, SDF, and other file formats to PyBullet. In addition to physics simulation, Pybullet includes rendering, including virtual reality headset compatibility, a CPU renderer, and OpenGL visualization [39]. Additionally, this platform is used in different RL research applications such as teaching manipulators to pick or throw objects [40,41], adjusting and enhancing MuJoCo robot control [42], and imitating motions for humans [43] or

animals [44–46], by robots. On the downside, this platform requires a large effort to create a fully detailed environment (i.e., everything needs to be created from scratch on Gazebo and imported into PyBullet) [47] and does not have enough environment resources on websites.

DART: This collaborative, cross-platform, open-source library called DART (Dynamic Animation and Robotics Toolkit) was created by the Georgia Institute of Technology's Graphics Lab and Humanoid Robotics Lab with ongoing assistance from the Personal Robotics Lab at the University of Washington and the Open-Source Robotics Foundation. This platform can be handled by two programming languages (i.e., C++ and Python), and can be integrated with the Gazebo platform [38] to create a different environment and import it as URDF or SDF files into DART. As a result of its multibody dynamic simulator and different kinematic tools for control and motion planning, DART has applications in robotics and computer animation [48]. Furthermore, it is used in RL research work like enhancing RL algorithms for different robots and tasks (e.g., robotic arm simulator, and hexapod locomotion in [49] (or) cart-pole swing-up, the double inverted pendulum, locomotion of a hopper, and block-throwing of a manipulator in [50] (or) pendubot swing-up, and hexapod robot moving in [51] (or) legged robot hoping or bipedal moving in [52]), overcoming the challenges that may face the robots during learning (e.g., the damage that may be done for various robots (i.e., wheeled robot, and six-legged robot) [53], and the falling that may be done [54]), and learning different skills (e.g., navigate cloth [55], assist dressing [56]). On the other hand, the DART platform requires a high experience and skills to create the environment details, because it was also created from scratch on Gazebo and imported into DART (similar to PyBullet) [47]. In addition, it doesn't have enough environmental resources on various online websites, and there is a lot of required enhancement for the available communication protocols.

MuJoCo: a unique physics engine called MuJoCo (Multi-Joint Dynamics with Contact) allows for accurate, effective rigid body simulations with contacts. It supports three different programming languages (i.e., C, C++, and Python); and it also can be integrated with Python OpenAI Gym, to perform RL algorithms enhancement, on the ready environments of OpenAI Gym. In addition, environments with continuous control tasks, like walking or running, may be made with MuJoCo [57]. Unfortunately, due to the environment creation difficulty, most of the RL work that is based on this platform, is proposed by using the ready environment. Thus, multiple MuJoCo ready environments have been used to evaluate a variety of RL algorithms (e.g., HalfCheetah, Hopper, Walker2d, and Ant in [58,59]). Therefore, the MuJoCo platform doesn't have efficient online website resources.

RaiSim: a closed-source multi-body physics engine platform based mainly on C++ programming language, that can be used through different operating systems (e.g., Linux, and Windows) [60]. This platform is used in some applications, by using ANYmal ready locomotion model, like RL algorithms implementation and developments (e.g., motion path optimization learning by importance sampling [61], and moving control in non-flat space by terrain-aware locomotion [62,63]) (or) imitate the human and animals' behaviors (e.g., recover from falling in hard environments [64], and learning the dexterous circus abilities [65]). Furthermore, it works very accurately and quickly, with respect to other platforms, in some ready environments (e.g., speed control, momentum, and energy tests for ANYmal robot) or simple environments that are created from scratch (e.g., friction rolling test, bouncing ball due to elasticity test, hard contact test of many balls) [66]. Although, it includes the ability to add XML or URDF files for sensors or bodies; it is difficult to find resources for this platform because it is a closed-source, and the creation of detailed agents or environments is difficult, that is the reason for using the ready robot models (e.g., ANYmal locomotion) for the most of applications. In addition, it is not suitable for drone applications, because it doesn't have significant libraries for that [67].

Isaac: a closed-source NVIDIA robotics platform based on the PhysX physics engine and Linux operating system, that can be programmed by C or Python programming language, to create virtual environments [68]. Due to its high capability to make simulation environments with a high degree of similarity to the real world, it's mainly used in synthetic

data generation for various computer vision tasks (e.g., classification, object detection, semantic segmentation, and depth estimation) [69,70]. It is accepted the URDF files and contains some extensions (e.g., sensors, control, and RL algorithms). On the other hand, this platform consumes a huge memory size during simulation, with respect to other platforms, and gets a lot of failures through performing tasks and algorithms (i.e., pick and place tasks, and throw tasks) [69]. However, there is some previous trials to integrate some libraries to enhance the implementation and development of the algorithms on this platform, but this integration for a little number of environments (e.g., Ant, Humanoid, Franka-cube-stack, Ingenuity, ShadowHand, ANYmal, Allegro, and TriFinger); without any flexibility in environment creation through this integration (Isaac Gym or OpenAI Gym) [71,72].

AirSim: an open-source simulation platform called Aerial Informatics and Robotics Simulation (AirSim), created especially for autonomous vehicle (e.g., drone, and car) research and developments by Microsoft [73]. This platform is based on the Unreal Engine for environment creation (recently, there is an experimental version that is based on Unity), because it has a high capability of realistic environment creation easily from a high programming layer level (i.e., the AirSim is just a plugin put inside Unreal environment). In addition, it can be programmed by different programming languages (i.e., C++, Python, C#, and Java) [73]. However, a limited number of environments, are released (e.g., Urban, CityEnviron, and Woodland environments) [74]; and contain only two types of drones [75]. These two limitation features are the reason for using this platform mainly in algorithms implementation and developments only (e.g., obstacle avoidance algorithms [74,76], object detection and tracking algorithm [75], swarm path planning algorithm [75], autonomous driving algorithm [75], and moving algorithms in a non-stationary environment [75]). Also, it doesn't have enough online resources for adjusting the environments or the drones. On the other hand, there aresome trials to overcome this platform's drawbacks by adding a programmable engine called PEDRA, which includes some additional environments for RL drone applications. It represents an abstraction layer between the low-level Python programming language, and the user interface; to reduce the time and the effort of users during algorithms implementation and environment creation. In other words, the user interfaces with a configuration file (.cfg), to specify the environments with algorithm options (e.g., one drone with vanilla RL algorithm, or prioritized experience relay with deep duel Q-learning) [77,78]. But, these trials are not sufficient enough; because it focuses on reducing the difficulties for the users' platform, not increasing the platform capability (i.e., drone design creation); and the PEDRA doesn't have enough online resources too. So, the limitation problem for the AirSim platform still exists.

**Table 1.** Comparison between the most popular RL existing platforms in terms of programming language, communication protocol, and environment creation; with respect to the proposed one.

| Simulation Platform | Programming Language | Communication Protocol | Environment Creation | Comment |
|---|---|---|---|---|
| OpenAI Gym [25] | Python | ___ | Very Complex | So limited environments, without low possibility for creation |
| Unity ML-Agents [33] | C# | ___ | Easy | Hard programing language, and very hard to adding or developing algorithms |
| PyBullet [39] | Python | TCP (or) UDP | Complex | Complexity of environment creation on Gazebo, is high [47] |
| DART [48] | C++ (or) Python | UDP | Complex | Complexity of environment creation on Gazebo, is high [47]. Also, the communication protocol need to be enhanced |

**Table 1.** *Cont.*

| Simulation Platform | Programming Language | Communication Protocol | Environment Creation | Comment |
|---|---|---|---|---|
| MuJoCo [57] | C (or) C++ (or) Python | ___ | Complex | There is a limited number of environment, and the environment creation is very complicated |
| RaiSim [60] | C++ | ___ | Complex | Complexity of environment creation. In addition, doesn't have suitable libraries for aerial applications [67] |
| Isaac [68] | C (or) Python | ___ | Medium Complex | Not suitable for algorithms implementation and developments, due to failure cases [69] |
| AirSim [73] | C++ (or) Python (or) C++ (or) Java | ___ | Complex | There is limited number of available environments, and only two drones' type [73,75] |
| Proposed Work | Python | UDP | Easy | The online resources is widely spread, the algorithms implementation and development is allowable, and suitable for infinite number of applications |

#### 1.1.2. Communication Protocol

From previous work that is mentioned in (Table 1), the two possible communication protocols for the proposed framework are: (1) Transmission Control Protocol (TCP), and (2) User Datagram Protocol (UDP). Both protocols are similar because they are examples of transport layer protocols. But, the UDP is a connectionless protocol, and TCP is a connection-oriented protocol. This indicates that whereas UDP does not require any connection before communicating, TCP does. In addition, the acknowledgment mechanism is used by TCP, but not used in UDP. This process involves the sender receiving an acknowledgment from the recipient and determining whether it is positive or negative. Also, while UDP does not adhere to the flow control technique (i.e., the package sending slices, slices ordering) used by TCP to ensure that many packets are not transmitted to the recipient at once, the latter does. The connection-oriented, error checking and flow control of the TCP, reduce its speed of data transfer with respect to the UDP. Finally, the UDP can be established between one python server and multi-agents, but the TCP can't [79,80].

There is a very important note, which is demonstrated by making some experimental tests of using each one (i.e., TCP, and UDP) during the task of implementing an elevation control on a quadcopter drone. Mainly, the UDP is very faster and more robust than the TCP, because the TCP, in most simulation cases, enters an infinite loop due to making some simple errors during dividing the data package into slices and sending. Herby, the proposed platform is based on UDP instead of TCP.

### *1.2. Contributions and Proposed Approach*

The proposed framework (Figure 1) is based on three main components: (1) Unity simulation software, to create the RL environment; (2) Python programming language, to create the RL agent; and (3) The UDP communication protocol, to perform interaction between RL agent and its environment. Hereby, the existing platforms based on these

three components are explained in the next sections. Also, for reliability and applicability proofing, two popular RL algorithms (Vanilla Policy Gradient (VPG) [81] and Actor-Critic (A2C) [82]) are implemented, to control the elevation of a quadcopter drone.

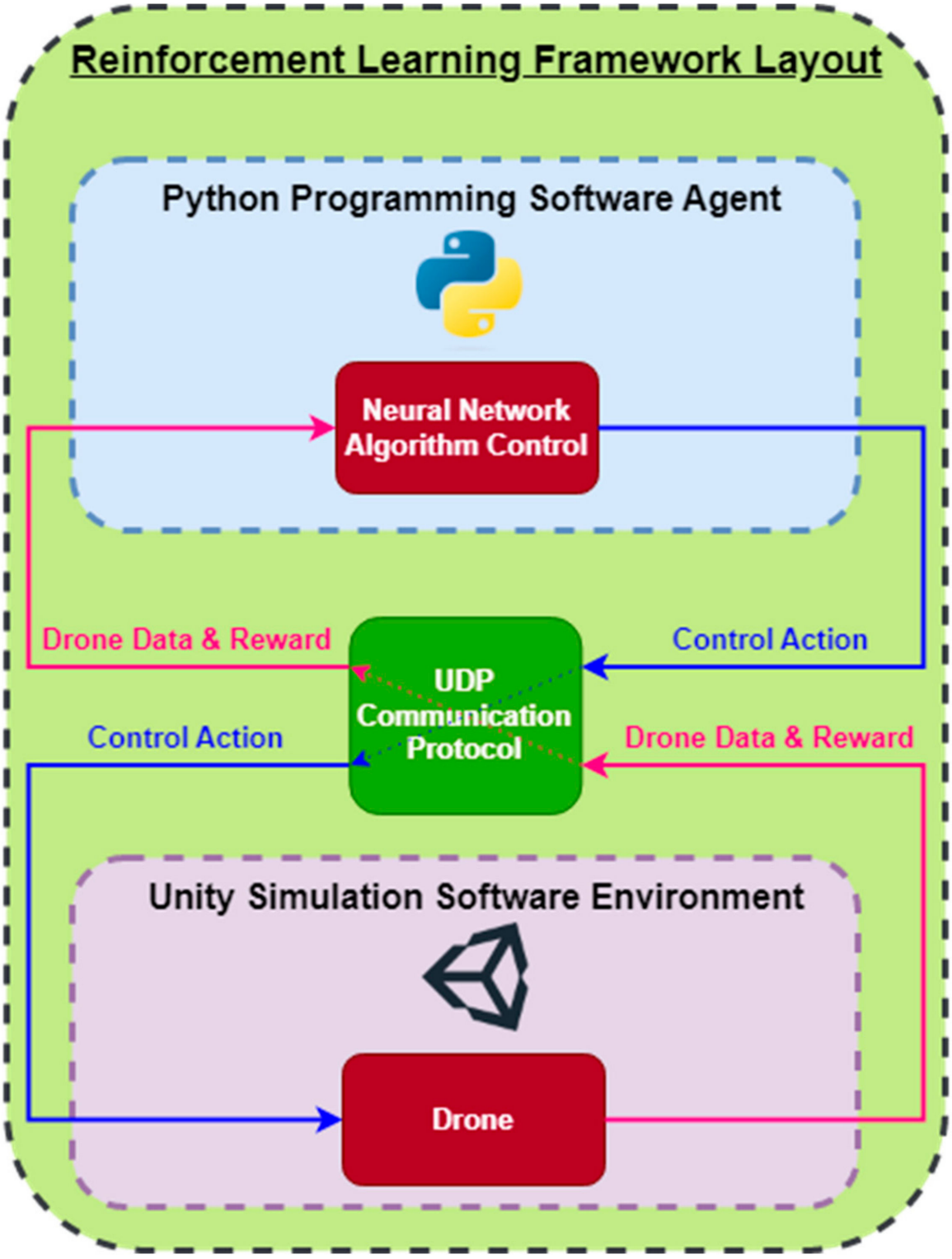


**Figure 1.** The proposed framework layout (the interaction between Python agent and Unity environment by using the UDP communication protocol).

Hereby, the paper's contribution can be stated in: (1) proposing a robust, flexible, scalable RL framework, based on a UDP interaction between Python RL agent, and Unity environment; (2) implementing the proposed framework on the quadcopter drone elevation control task, and making a comparison between VPG and A2C algorithms performance on the same task.

This paper contains four sections, besides the introduction section which represents Section 1. In addition, Section 2 is the methodology of the proposed framework and implementation details, Section 3 is the results and comparisons of experiments on the quadcopter elevation control using RL algorithms, Section 4 is limitations and future work for the proposed work, and Section 5 discusses the conclusions of this study.

## 2. Methodology

In this section, a clear and comprehensive explanation of the proposed framework and the basics for it, is demonstrated. In addition, the validation for this framework and proof for its applicability, is illustrated too, by applying a comparison between two of the most significant reinforcement algorithms (i.e., Vanilla Policy Gradient (VPG), and Actor-Critic (A2C)) on a control problem for the drone elevation.

### *2.1. The Proposed Framework*

The integration between a flexible programing language (i.e., Python), and powerful simulation software (i.e., unity); is the backbone for the proposed framework (Figures 2 and 3) This integration makes full use of both, to overcome the drawbacks of each one separately and increase the advantages for both. The framework depends on creating a reinforcement learning (RL) agent by using the Python programing language, and creating a simulation environment in unity software. The agent interacts with the environment through the UDP communication protocol. The interaction is done by producing an action from the core of the agent (i.e., the neural network algorithm), to be performed on the environment, and monitoring the response updates (i.e., the data and reward) of the environment core (i.e., the drone) to be used as the inputs for the agent in next time step.

The UDP communication protocol implementation is done by creating a UDP server and a UDP client; by using a computer with a specification of (12th Gen Intel(R) Core(TM) i7-12700K 3.60 GHz) processor, (32.0 GB) RAM, and (NVIDIA GeForce RTX 3090) GPU. Firstly, the UDP server contains five steps: (1) Create a UDP Python socket, (2) Bind the socket to the specified server address, (3) Wait for the environment datagram packet arriving from the C# client, (4) Process the environment datagram packet and send the agent action reply to the client, and (5) Go back to Step 3. On the other hand, the UDP Client contains five steps: (1) Create a UDP C# socket, (2) Bind the socket to the specified client address, (3) Send the environment data message to the Python server, (4) Wait until the agent action response from the Python server is received, (5) Go back to step 3. This implementation is an enhanced version of a simple one of sending and receiving strings that created by Youssef Elashry, to solve the common problem of two-way communication between Python and Unity [83]. The enhancement is achieved by making the process of communication able to read the game object data from Unity and transfer an array of data (e.g., drone data, and reward) between the Unity simulation environment and the Python agent through running simulation, to perform the reinforcement learning concept, instead of transfer stings only. In addition, this two-way communication between Python and Unity, is based on sending and receiving bytes. So, all messages (i.e., environment data and agent action) are converted to bytes through sending and receiving, then converted to their original case again before using it.

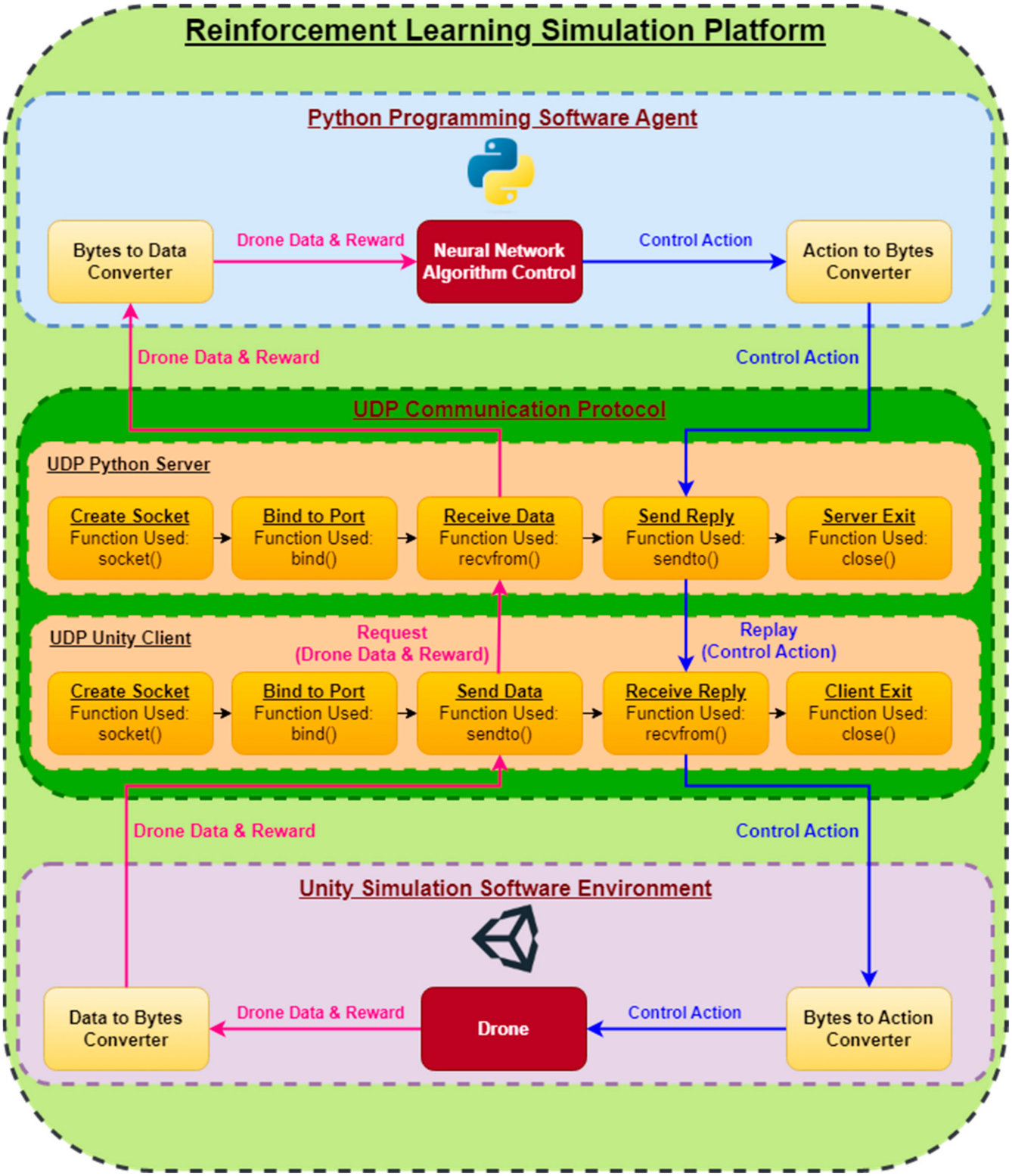


**Figure 2.** Overall explanation for the proposed framework.

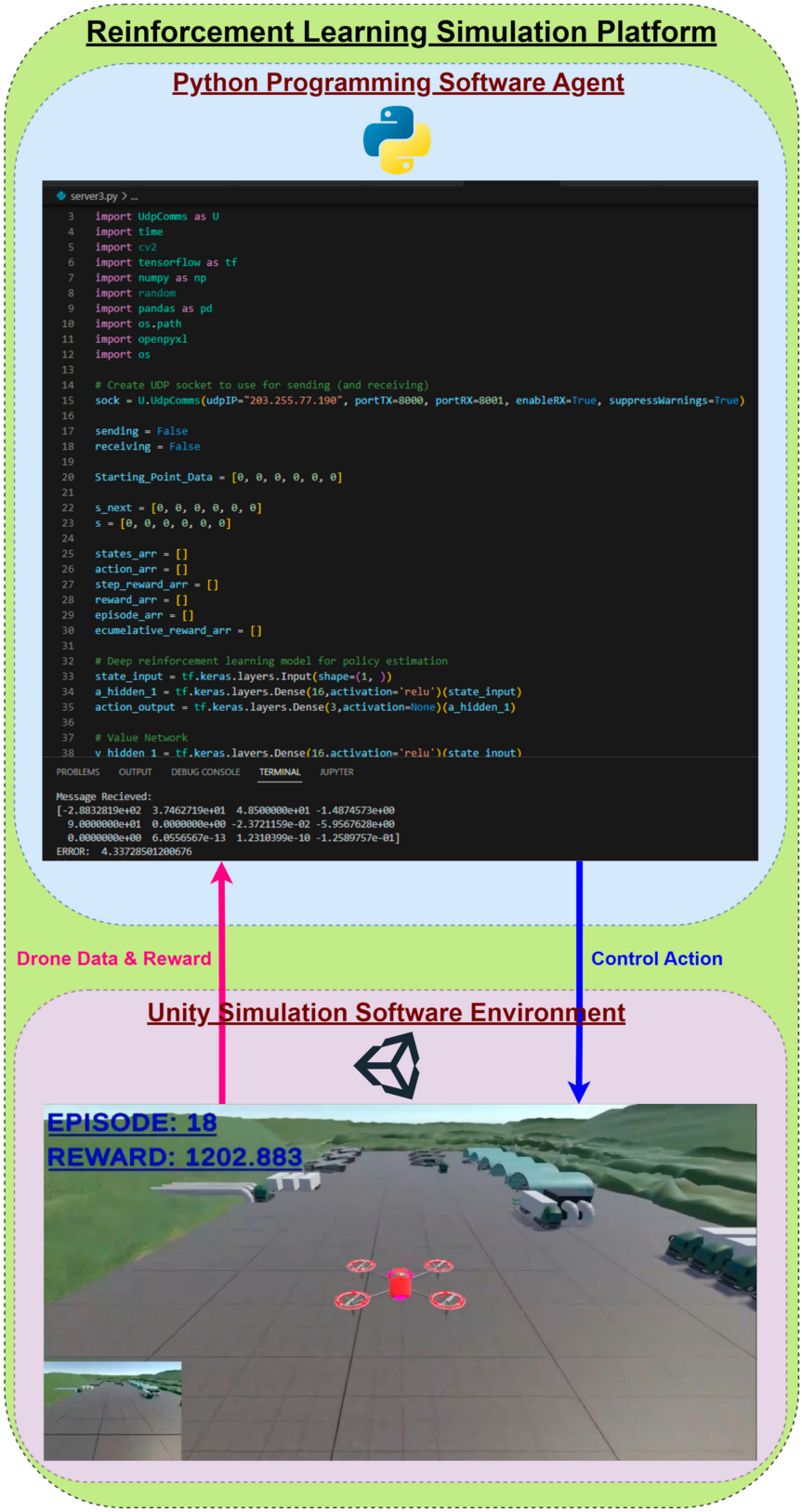


**Figure 3.** Overall explanation for the proposed framework after implementation.

### *2.2. The Drone*

The game object in the Unity environment is a quadcopter robot (Figure 4), that is imported from Unity Asset Store [84]. Also, a complete airport surrounding is added, for just visualization enhancement, from Sketchfab's leading platform for 3D and AR environments on the web [85]. The quadcopter represents a six degree of freedom robot, because it has three rotation motions (ψ, ∅, and Θ), three transitional motions (x, y, and z); and only four distinct inputs represented by rotor thrust forces (F1, F2, F3, and F4).

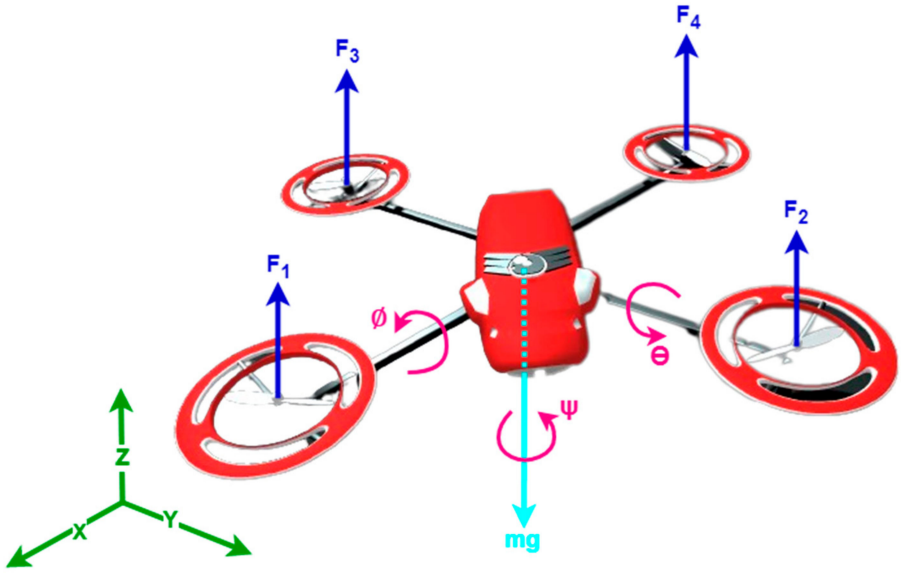


**Figure 4.** The quadcopter coordinate frame.

The quadcopter parameters are identified as follows: (1) the frame structure is a stiff symmetric body with a 2-m square distance, (2) the mass (m) is 20 Kg concentrated in the center of gravity and the geometrical center, (3) the gravitational acceleration (g) is 9.8 m/s$^2$, (4) the maximum thrust force per propeller is 100 N.

The validation case study is the quadcopter elevation control (Figure 5), which depends on comparing the desired quadcopter elevation (Y) with the environment elevation (Y') to produce an error signal (E), that is used as input for the neural network agent, to predict the required action and send it to the quadcopter environment. The generated action from the RL agent is the required change for the thrust force to achieve the target elevation (i.e., all forces from propellers are increased or decreased by the same value that is produced by the agent).

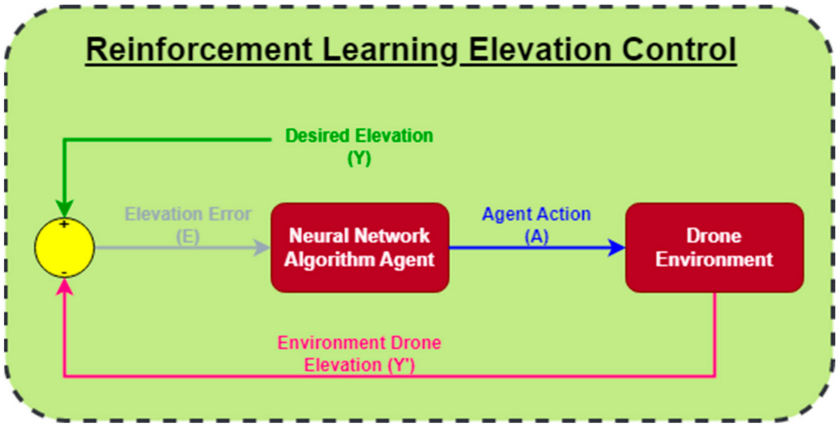


**Figure 5.** Control layout for the quadcopter elevation.

### *2.3. Reinforcement Learning (RL) Agent Algorithms*

The first implemented algorithm is the VPG (Algorithm 1). Policy gradients' main principle is to increase the probability of actions that result in greater returns and decrease the probabilities of actions that result in lower returns until the algorithm reaches the optimal action to take. Furthermore, this algorithm is an on-policy, that can be implemented as producing discrete or continuous action on environment spaces. Hereby, a stochastic policy is trained using a simple policy gradient in an on-policy manner. This indicates that it samples activities during exploration in accordance with the most recent iteration of its stochastic strategy. Action selection's level of randomness is influenced by both the training process and the starting conditions. As the update rule encourages the policy to take advantage of rewards that it has previously discovered, the policy normally becomes gradually less random throughout training. The policy could become stuck in local optima as a result of this [86].

The VPG agent depends on selecting an action (a) by the distribution probability (P) based on policy ($\pi$). In other words, the distribution probability of action ($P(a \mid \pi_\theta(s))$) depends on the current state (s) for the policy $\pi_\theta(s)$, that contains the parameter $\theta$. In other words, the observation state (s) that represent the elevation error (E) in the control task, inputs to a fully connected neural network model that represents $\pi_\theta(s)$ in the VPG algorithm, and then the neural network model produces the required thrust action (i.e., increasing or decreasing) to eliminate the elevation error. For each episode, the expected return ($E_{\pi\theta}[R]$) is computed using (Equation (1)), where R is the cumulative reward. In addition, a gradient descent (Equation (2)) method with learning rate ($\alpha$), is used to optimize the parameter $\theta$ that maximizes (E). As seen in Equations (1) and (2), the probability value for selecting action (a) in the state (s) is increased when the result of expected reward (R) is positive, by changing the ($\theta$) value to increase this probability. On the other hand, it is possible to reduce the probability of taking action (a) in the state (s), by making the expected reward (R) negative which affects the ($\theta$) value to decrease this action probability. The neural network contains only one hidden layer with 16 neurons, and each neuron contains an activation function called Rectified Linear Unit (ReLU). In addition, the selected optimizer is called the Adam optimizer with a learning rate of 0.001.

$$E_{\pi\theta}[R] = \int_a R(a) \cdot P(a|\pi_\theta(s)) \tag{1}$$

$$\theta_{t+1} = \theta_t + \alpha R \cdot \nabla_\theta \log(P(a|\pi_\theta(s))) \tag{2}$$

**Algorithm 1**. Vanilla Policy Gradient (Pseudocode)

Randomly initialize policy network trainable parameter $\theta$
**for** $t = 1 : T$ **do**
1. Collect a set of drone data by applying the current policy
2. At each time step $t$ in each trajectory, compute the return (R) $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$
3. Update the policy, using a policy gradient method $\theta_{t+1} = \theta_t + \nabla_\theta \, log \, P(a_t \mid \pi_\theta(s_t))$
4. Insert the policy gradient output into ADAM optimizer

**end for**

The Actor-Critic (A2C), which is based on the policy model and the value function, is the second algorithm that has been put into use (Algorithm 2). Learning the value function in addition to the policy (the Actor updates the policy parameters, and the Critic updates the value function parameters) makes a lot of sense since doing so will help with updating the policy. So, the A2C technique reduces gradient variance in VPGs [82].

For the implementation of the A2C algorithm, the action value function (Q(s,a)) is separated into two parts: (1) the potential value (V(s)) that is independent of action (a), and (2) the advantage (equation 3) (A(a,s)) that depends on action (a) and state (s), where

($r_t$) is the current reward and ($\gamma$) is the discount factor. Hereby, a value function (V(s)) is constructed and optimized based on the estimation value in that state, and the policy gradient is done on the advantage function (A(a,s)) to optimize the action selection in that state (i.e., two neural networks are created, one for the value estimation, and the other to the policy function that based on advantage function for action prediction). In other words, the policy gradient is applied to the advantage function (A(a,s)) to minimize it, instead of maximizing the expected return ($E_{\pi\theta}[R]$) in the VPG algorithm. So, a more stable conversion is expected (i.e., when the reward increases significantly, the value will increase in this scenario, and the value loss will therefore be greater than policy loss) Finally, there are two neural networks (one for predicting the state values, and the other for the required action prediction), each one is similar to the previous one that used with VPG (i.e., contains only one hidden layer with 16 neurons, each neuron contain ReLU activation function. In addition, the selected optimizer is Adam with a 0.001 learning rate. The tuning process for all used neural networks is done manually, based on the author's experience [87,88], to overcome common problems like overfitting and underfitting problems.

$$A(a_t, s_t) = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3}$$

**Algorithm 2**. Actor-Critic Algorithm (Pseudocode)

Randomly initialize networks trainable parameters $w$ for value and $\theta$ for policy

**for** $t = 1 : T$ **do**

1. Apply an action $a \sim \pi_\theta(a|s)$
2. Collect the data of reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
3. Update the policy network parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta ln\ \pi_\theta(a|s)$
4. Calculate the advantage at time t
5. Compute the correction (TD error) for action-value at time t:
   $A_t = r_t + \gamma V_w(s',a') - V_w(s,a)$
   and use it to update the parameters of value network:
   $w \leftarrow w + \alpha_w A_t \nabla_w V_w(s,a)$
6. Insert the policy gradient output into ADAM optimizer
7. Update $a \leftarrow a'$ and $s \leftarrow s'$

**end for**

## 3. Results and Discussion

The main target for the results section is the validation for the proposed integrated framework to be used in reinforcement learning (RL) problems. Furthermore, the validation is an essential step, to prove the framework's flexibility, scalability, and robustness. The flexibility, scalability, and robustness are proven by implementing two RL algorithms (i.e., Vanilla Policy Gradient (VPG) algorithm and Actor-Critic (A2C) algorithm) using the Python programming language. In addition, creating a custom environment for drones by using Unity simulation software.

The evaluation process for the RL algorithms depends on many factors, that are based on collecting positive rewards by the agent as much as he can. These positive rewards mean that the agent learning process, is done as required. The evaluation metrics such as total reward, average reward, and cumulative reward; are all important during algorithm training, because the target is increasing the reward in the short-run (represented by total reward and average reward) and the long-run (represented by cumulative reward) also.

The reward curves shapes in Figure 6 show that the VPG algorithm learning curve fluctuated higher than the A2C algorithm curve; because the first one is much more simple than the second one. On the other hand, the A2C algorithm curve takes much more episodes to converge and outperform the VPG algorithm, because the algorithm is based on making optimization for two neural networks (i.e., policy neural network and value neural network). so, it needs much more time but can reach to higher average reward with minimum fluctuation.

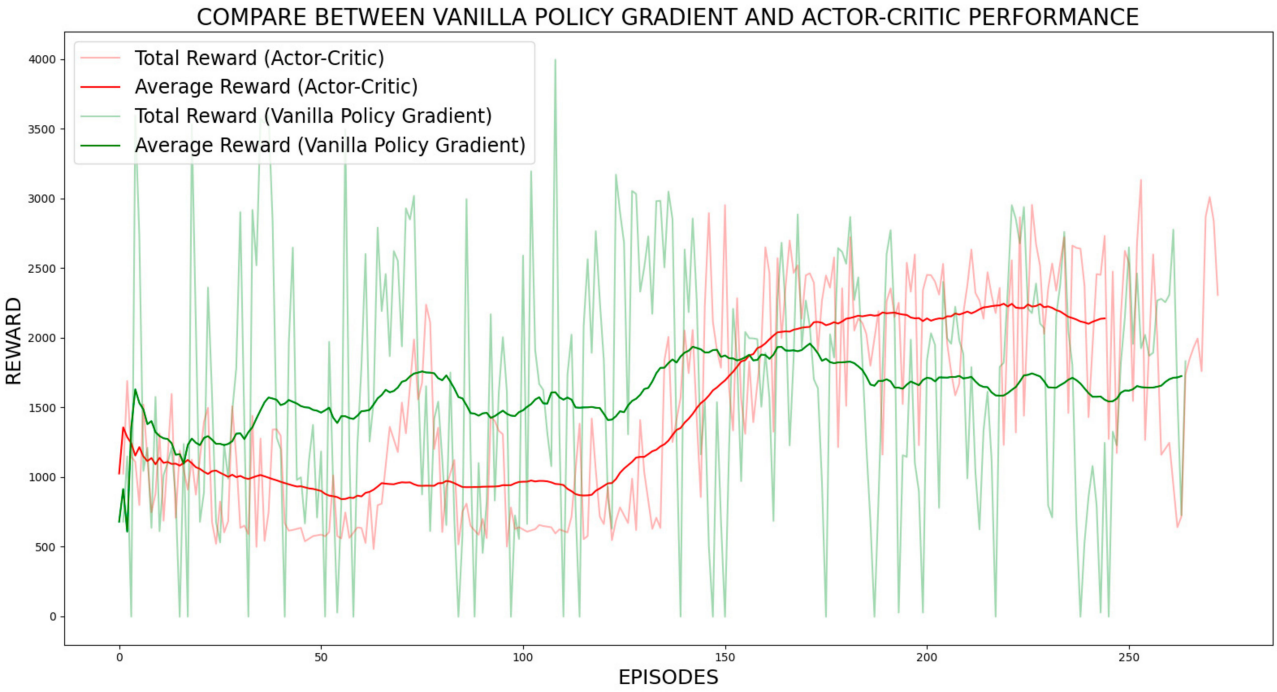


**Figure 6.** Total reward and average reward curves for Vanilla Policy Gradient (VPG) and Actor-Critic (A2C) algorithms (A2C converge slower than VPG, with much more stability).

In addition, the cumulative reward curves in Figure 7 prove the ability of both algorithms to get a high reward in the long-run. Also, the A2C algorithm can overcome the delay in converges and the curve be closer to the VPG algorithm curve in long-run, because it can collect a higher average reward in the short-run.

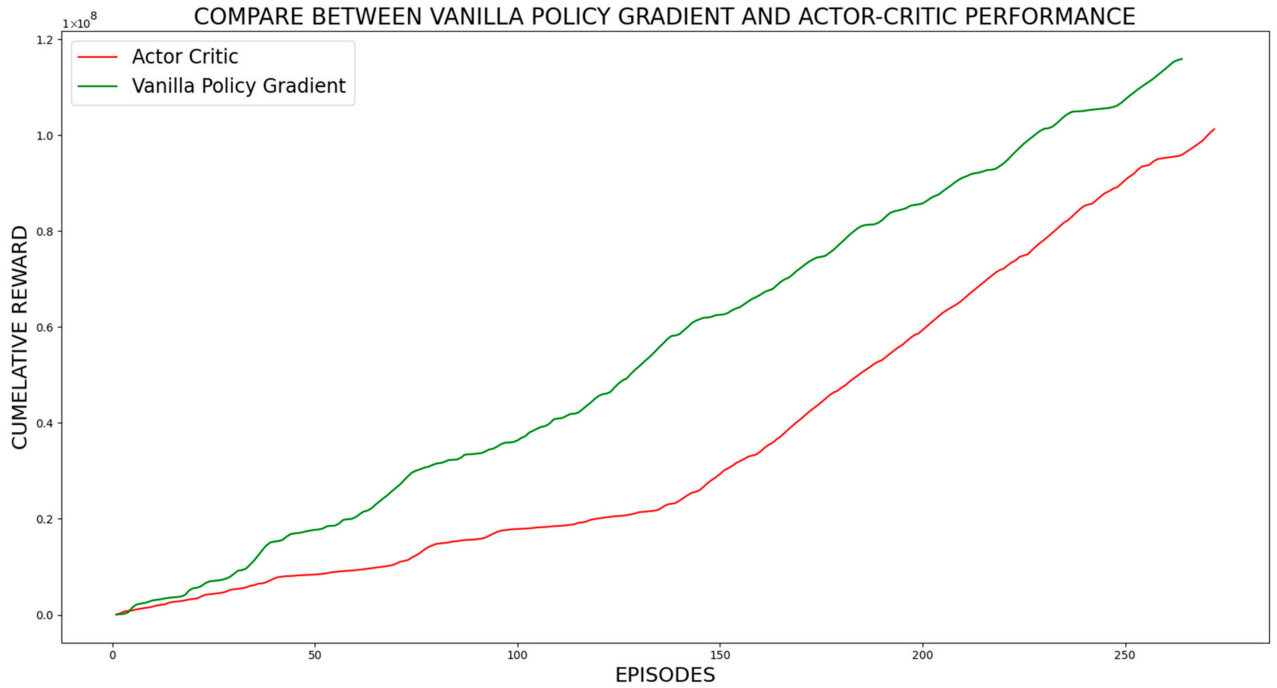


**Figure 7.** Cumulative reward curves for Vanilla Policy Gradient (VPG) and Actor-Critic (A2C) algorithms (A2C outperform the VPG on the long run).

Also, the comparison of error signals that performing due to each algorithm in Figure 8, demonstrates the compensation of error signal. But, the A2C algorithm takes a longer time than VPG, to get the best solution to converge with higher performance and lower fluctuation. In other words, the A2C algorithm has higher error signal fluctuation, at the beginning of training steps. But, in long run, the algorithm can perform more stable than the VPG algorithm. Even with large error values, the algorithm gets the required elevation quickly.

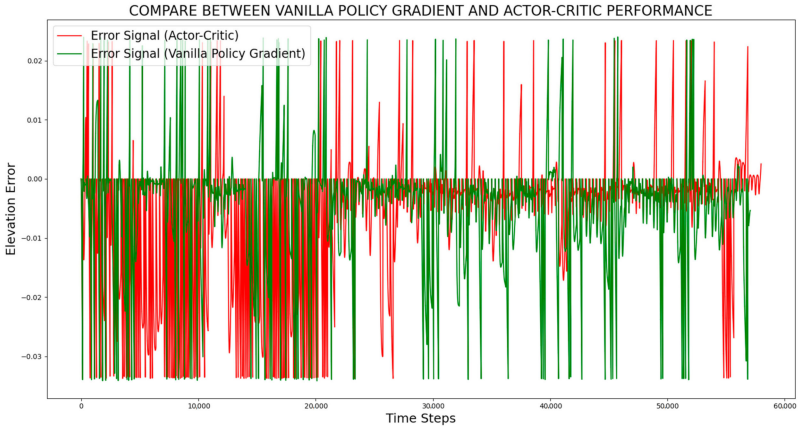


**Figure 8.** Elevation error signal for Vanilla Policy Gradient (VPG) and Actor-Critic (A2C) algorithms (A2C compensate the error with higher stability than VPG, but after much more time steps).

Finally, the proposed integrated simulation framework can be used as a simulation platform for RL problems. Because of the ability to create infinite numbers of environments and implement different agent algorithms (the simple and complicated algorithms, and it gets an expected performance for the implemented algorithms, that declared when getting high performance from A2C algorithm with respect to VPG algorithm). Furthermore, the resultant video files of the training record for the platform environment running and RL agent training, are uploaded in these video links [89–91].

## 4. Limitations and Future Work

In the previous section, the declaration for the applicability of the proposed framework is done, and the results of the platform validation process are explained in detail. Hereby, in this section, the limitations and future work for the proposed framework are discussed based on the expected impacts on the research community. The impacts of the proposed framework on the research field, can be affected on two application fields: (1) reinforcement learning (RL) applications field, and (2) computer vision applications field.

Firstly, for the field of RL, the proposed framework will be the basis for a flexible, robust, and scalable platform. This platform will be able to simulate any complex environment easily by using the Unity simulation engine software, and develop any RL algorithm by implementing it easily on the Python server. This type of platform overcomes the drawbacks of implementing some RL applications (e.g., making an RL learning process for drones in a real-world environment, because the crashing probability is very high for the drone or the surrounding real-world environment). So, an infinite number of RL applications can be performed through the design stage on this simulation platform, and then take a risk of real-life implementation. In addition, the extension for the RL platform to cover the multi-agent applications, is so important, to simulate the interaction between agents in one environment during the RL training, which is hard to make the training process from scratch in the real-world.

Secondly, in the computer vision field, the proposed framework can be extended to generate synthetic data (the data generated through simulation, instead of real-world data), by creating an infinite number of the simulation environment. Each simulation environment in the Unity engine will contain a multi-virtual camera, to construct different

camera views. Then send each camera frame, by using the UDP communication protocol, to the Python server, which easily makes each frame processing, and finally saves each frame in a directory that contains the synthetic data. So, the platform can easily create an infinite number of image applications (e.g., a semantic segmented, object tracked, and classification).

## 5. Conclusions

The main contribution of this paper is the creation of a high-performance platform to be used in reinforcement learning (RL) applications, based on the proposed framework. This contribution is done by passing through many steps: (1) makes a clear comparison of the existing tools that can be used in RL applications, (2) puts a layout for the framework and then explained it in detail, (3) validates the framework by implementing it on a quadcopter drone control using RL algorithms, and finally (4) explains the expectations and impacts of creating such platforms on the research field.

Firstly, this paper started by discussing the most recent challenges in the RL field, to create the most significant simulation platforms, that can be used as a preprocess for creating a pre-trained RL agent, due to the risk of making the agent training in the real-world directly. This discussion makes a complete viewpoint of the importance of creating a flexible, robust, and scalable framework; to be the basis of creating the most complementary RL simulation platform.

Secondly, the proposed framework is based on creating a UDP communication protocol between the Python programming language (as the RL agent), and the Unity simulation software (as the RL environment). To be used as a platform for such applications.

Thirdly, the proof for the framework's applicability to the RL problems, is done by implementing it on the elevation control of a quadcopter drone; and using two of the most popular RL algorithms (i.e., Vanilla Policy Gradient (VPG), and Actor-Critic (A2C)). To declare the flexibility, robustness, and scalability of the proposed work, in the creation and development of RL algorithms.

Finally, the expected impacts, limitations, and future work of this research, are mentioned as the effects on two fields (RL, and computer vision) by the applicability of creating a complementary platform to be used in the RL applications, and the synthetic data generation for the computer vision application.

**Author Contributions:** Conceptualization, M.A.B.A.; methodology, M.A.B.A.; software, M.A.B.A.; validation, M.A.B.A.; formal analysis, M.A.B.A.; investigation, M.A.B.A.; resources, M.A.B.A.; data curation, M.A.B.A.; writing—original draft preparation, M.A.B.A.; writing—review and editing, M.A.B.A.; visualization, M.A.B.A.; supervision, H.-S.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the MSIT (Ministry of Science and ICT), Korea, under the Grand Information Technology Research Center support program (IITP-2023-2020-0-01462), supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Acknowledgments:** This work was supported by the MSIT (Ministry of Science and ICT), Korea, under the Grand Information Technology Research Center support program (IITP-2023-2020-0- 01462), supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

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

## Nomenclature

| Symbol | Description | Units |
|---|---|---|
| a | Produced Action from Agent | — |
| A(s,a) | The Advantage Function | — |
| A2C | Actor-Critic | — |
| AI | Artificial Intelligence | — |
| E | Error Signal for Quadcopter Elevation | Meter (m) |
| Eπθ[R] | Expected Return from Environment | — |
| F1 | Quadcopter Thrust Force from Motor 1 | Newton (N) |
| F2 | Quadcopter Thrust Force from Motor 2 | Newton (N) |
| F3 | Quadcopter Thrust Force from Motor 3 | Newton (N) |
| F4 | Quadcopter Thrust Force from Motor 4 | Newton (N) |
| g | Gravitational Acceleration | Meter/Square Second (m/s2) |
| IP | Internet Protocol | — |
| m | Quadcopter Mass | Kilogram (Kg) |
| ML | Machine Learning | — |
| NN | Neural Network | — |
| P | Action Probability Distribution | — |
| P(a \| πθ(s)) | Probability of Action Based on Specific Policy | — |
| Q(s,a) | The Action Value Function | — |
| r | The Current Step Reward from Environment | — |
| R(a) | Return from Environment | — |
| RL | Reinforcement Learning | — |
| s | Quadcopter States Data | Meter (m) |
| TCP | Transmission Control Protocol | — |
| UDP | User Datagram Protocol | — |
| V(s) | The State Value Function | — |
| VPG | Vanilla Policy Gradient | — |
| w | Trainable Parameters for Value Neural Network | — |
| X | Quadcopter Position in X Axis | Meter (m) |
| Y | Quadcopter Position in Y Axis | Meter (m) |
| Y′ | The Measured Quadcopter Position in Y Axis from The Unity Environment | Meter (m) |
| Z | Quadcopter Position in Z Axis | Meter (m) |
| ∅ | Quadcopter Rotation Angle Around X Axis | Degree |
| α | Learning Rate for The Neural Network | — |
| γ | The Discount Factor for Total Return | — |
| θ | Trainable Parameters for Policy Neural Network | — |
| πθ(s) | The Policy Between States and Actions | — |
| ψ | Quadcopter Rotation Angle Around Z Axis | Degree |
| Θ | Quadcopter Rotation Angle Around Y Axis | Degree |

## References

1. Li, Y. Reinforcement Learning Applications. *arXiv* **2019**, arXiv:1908.06973. [CrossRef]
2. Norgeot, B.; Glicksberg, B.S.; Butte, A.J. A call for deep-learning healthcare. *Nat. Med.* **2019**, *25*, 14–15. [CrossRef]
3. Komorowski, M.; Celi, L.A.; Badawi, O.; Gordon, A.C.; Faisal, A.A. The Artificial Intelligence Clinician learns optimal treatment strategies for sepsis in intensive care. *Nat. Med.* **2018**, *24*, 1716–1720. [CrossRef]
4. Li, C.Y.; Liang, X.; Hu, Z.; Xing, E.P. Hybrid Retrieval-Generation Reinforced Agent for Medical Image Report Generation. In Proceedings of the 32nd International Conference on Neural Information Processing Systems 2018 (NIPS 2018), Montréal, Canada, 3–8 December 2018; pp. 1537–1547.
5. Ling, Y.; Hasan, S.A.; Datla, V.; Qadir, A.; Lee, K.; Liu, J.; Farri, O. Diagnostic Inferencing via Improving Clinical Concept Extraction with Deep Reinforcement Learning: A Preliminary Study. In Proceedings of the 2nd Machine Learning for Healthcare Conference (PMLR), Boston, MA, USA, 18–19 August 2017; Volume 68, pp. 271–285.
6. Peng, Y.-S.; Tang, K.-F.; Lin, H.-T.; Chang, E.Y. REFUEL: Exploring Sparse Features in Deep Reinforcement Learning for Fast Disease Diagnosis. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS 2018), Montréal, QC, Canada, 3–8 December 2018; pp. 7333–7342.

7. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **2018**, *362*, 1140–1144. [CrossRef] [PubMed]
8. Aytar, Y.; Pfaff, T.; Budden, D.; Paine, T.; Wang, Z.; Freitas, N. Playing hard exploration games by watching YouTube. In Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, QC, Canada, 3–8 December 2018.
9. García-Sánchez, P.; Georgios, N. Yannakakis and Julian Togelius: Artificial Intelligence and Games. *Genet. Program. Evolvable Mach.* **2018**, *20*, 143–145. [CrossRef]
10. Chen, L.; Chang, C.; Chen, Z.; Tan, B.; Gašić, M.; Yu, K. Policy Adaptation for Deep Reinforcement Learning-Based Dialogue Management. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15 April 2018; pp. 6074–6078.
11. Hudson, D.A.; Manning, C.D. Compositional Attention Networks for Machine Reasoning. In Proceedings of the International Conference on Learning Representations) ICLR, Vancouver, BC, Canada, 30 April–3 May 2018.
12. Zhang, X.; Lapata, M. Sentence Simplification with Deep Reinforcement Learning. In Proceedings of the Empirical Methods in Natural Language Processing, Copenhagen, Denmark, 9–11 September 2017.
13. He, J.; Chen, J.; He, X.; Gao, J.; Li, L.; Deng, L.; Ostendorf, M. Deep Reinforcement Learning with a Natural Language Action Space. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, Berlin, Germany, 7–12 August 2016; pp. 1621–1630.
14. Li, D.; Zhao, D.; Zhang, Q.; Chen, Y. Reinforcement Learning and Deep Learning based Lateral Control for Autonomous Driving. *IEEE Comput. Intell.* **2018**, *14*. [CrossRef]
15. Pérez-Gil, Ó.; Barea, R.; López-Guillén, E.; Bergasa, L.M.; Gómez-Huélamo, C.; Gutiérrez, R.; Díaz-Díaz, A. Deep reinforcement learning based control for Autonomous Vehicles in CARLA. *Multimed. Tools Appl.* **2022**, *81*, 3553–3576. [CrossRef]
16. Lange, S.; Riedmiller, M.; Voigtländer, A. Autonomous Reinforcement Learning on Raw Visual Input Data in a Real world application. In Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, QLD, Australia, 10–15 June 2012.
17. O'Kelly, M.; Sinha, A.; Namkoong, H.; Duchi, J.; Tedrake, R. Scalable End-to-End Autonomous Vehicle Testing via Rare-event Simulation. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, QC, Canada, 3–8 December 2018; pp. 9849–9860.
18. Argall, B.D.; Chernova, S.; Veloso, M.; Browning, B. A survey of robot learning from demonstration. *Robot. Auton. Syst.* **2009**, *57*, 469–483. [CrossRef]
19. Kober, J.; Bagnell, J.A.; Peters, J. Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
20. Deisenroth, M.P. A Survey on Policy Search for Robotics. *Found. Trends Robot.* **2011**, *2*, 1–142. [CrossRef]
21. Zhu, Y.; Mottaghi, R.; Kolve, E.; Lim, J.J.; Gupta, A.; Fei-Fei, L.; Farhadi, A. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In Proceedings of the ICRA, Singapore, 16 September 2017.
22. Hwangbo, J.; Lee, J.; Dosovitskiy, A.; Bellicoso, D.; Tsounis, V.; Koltun, V.; Hutter, M. Learning agile and dynamic motor skills for legged robots. *Sci. Robot.* **2019**, *4*, eaau5872. [CrossRef]
23. Song, Y.; Steinweg, M.; Kaufmann, E.; Scaramuzza, D. Autonomous Drone Racing with Deep Reinforcement Learning. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021.
24. Hwangbo, J.; Sa, I.; Siegwart, R.; Hutter, M. Control of a Quadrotor with Reinforcement Learning. *IEEE Robot. Autom. Lett.* **2017**, *2*, 99. [CrossRef]
25. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym. *arXiv* **2016**, arXiv:1606.01540. [CrossRef]
26. Beysolow, T. *Applied Reinforcement Learning with Python With OpenAI Gym, Tensorf low, and Keras*; Apress Media LLC: San Francisco, CA, USA, 2019.
27. Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; Mordatch, I. Decision Transformer: Reinforcement Learning via Sequence Modeling. In Proceedings of the Thirty-Fifth Conference on Neural Information Processing Systems (NeurIPS), Virtual-Only, 6–14 December 2021.
28. Peng, X.B.; Kumar, A.; Zhang, G.; Levine, S. Advantage Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. In Proceedings of the International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, 26–30 April 2020.
29. Li, Y.-J.; Chang, H.-Y.; Lin, Y.-J.; Wu, P.-W.; Wang, Y.-C.F. Deep Reinforcement Learning for Playing 2.5D Fighting Games. In Proceedings of the 25th IEEE International Conference on Image Processing (ICIP), Athens, Greece, 7–10 October 2018.
30. Fujimoto, S.; Hoof, H.v.; Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. In Proceedings of the 35th International Conference on Machine Learning (ICML-18), Stockholmsmässan, Stockholm, Sweden, 10–15 July 2018.
31. Plappert, M.; Andrychowicz, M.; Ray, A.; McGrew, B.; Baker, B.; Powell, G.; Schneider, J.; Tobin, J.; Chociej, M.; Welinder, P.; et al. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *arXiv* **2018**, arXiv:1802.09464.
32. Matthew Hausknecht, P.S. Deep Recurrent Q-Learning for Partially Observable MDPs. In Proceedings of the Conference on ArtificialIntelligence (AAAI-15), Austin, TX, USA, 25–30 January 2015.

33. Majumder, A. *Deep Reinforcement Learning in Unity with Unity ML Toolkit*; Apress Media LLC: San Francisco, CA, USA, 2021.
34. Cao, Z.; Lin, C.-T. Reinforcement Learning from Hierarchical Critics. In Proceedings of the IEEE Transactions on Neural Networks and Learning Systems, Casablanca, Morocco, 14 May 2021.
35. Song, Y.; Wojcicki, A.; Lukasiewicz, T.; Wang, J.; Aryan, A.; Xu, Z.; Xu, M.; Ding, Z.; Wu, L. Arena: A General Evaluation Platform and Building Toolkit for Multi-Agent Intelligence. In Proceedings of the AAAI Conference on Artificial Intelligence: Multiagent Systems, Stanford, CA, USA, 21–23 March 2022; pp. 7253–7260.
36. Juliani, A.; Berges, V.-P.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; et al. Unity: A General Platform for Intelligent Agents. *arXiv* **2018**, arXiv:1809.02627.
37. Booth, J.; Booth, J. Marathon Environments: Multi-Agent Continuous Control Benchmarks in a Modern Video Game Engine. In Proceedings of the AAAI Workshop on Games and Simulations for Artificial Intelligence, Honolulu, HI, USA, 29 January 2019.
38. Koenig, N.; Howard, A. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sendai, Japan, 28 September–2 October 2004.
39. Coumans, E.; Bai, Y. PyBullet Quickstart Guide. 2016. Available online: https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit# (accessed on 14 March 2023).
40. Breyer, M.; Furrer, F.; Novkovic, T.; Siegwart, R.; Nieto, J. Comparing Task Simplifications to Learn Closed-Loop Object Picking Using Deep Reinforcement Learning. In Proceedings of the Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 25–29 October 2020.
41. Zeng, A.; Song, S.; Lee, J.; Rodriguez, A.; Funkhouser, T. TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. *IEEE Trans. Robot.* **2019**, *36*, 1307–1319. [CrossRef]
42. Choromanski, K.; Pacchiano, A.; Parker-Holder, J.; Tang, Y.; Jain, D.; Yang, Y.; Iscen, A.; Hsu, J.; Sindhwani, V. Provably Robust Blackbox Optimization for Reinforcement Learning. In Proceedings of the 3rd Conference on Robot Learning (CoRL), Osaka, Japan, 30 October–1 November 2019.
43. Peng, X.B.; Abbeel, P.; Levine, S.; van de Panne, M. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ACM Trans. Graph.* **2018**, *37*, 1–14. [CrossRef]
44. Peng, X.B.; Coumans, E.; Zhang, T.; Lee, T.-W.; Tan, J.; Levine, S. Learning Agile Robotic Locomotion Skills by Imitating Animals. In Proceedings of the Robotics: Science and Systems, Corvalis, OR, USA, 12–16 July 2020.
45. Singla, A.; Bhattacharya, S.; Dholakiya, D.; Bhatnagar, S.; Ghosal, A.; Amrutur, B.; Kolathaya, S. Realizing Learned Quadruped Locomotion Behaviors through Kinematic Motion Primitives. In Proceedings of the International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019.
46. Tan, J.; Zhang, T.; Coumans, E.; Iscen, A.; Bai, Y.; Hafner, D.; Bohez, S.; Vanhoucke, V. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. In Proceedings of the Robotics: Science and Systems, Pittsburgh, PA, USA, 26–30 June 2018.
47. Pyo, Y.; Cho, H.; Jung, R.; Lim, T. *ROS Robot Programming from the Basic Concept to Practical Programming and Robot Application*; ROBOTIS: Seoul, Republic of Korea, 2017.
48. Lee, J.; Grey, M.X.; Ha, S.; Kunz, T.; Jain, S.; Ye, Y.; Srinivasa, S.S.; Stilman, M.; Liu, C.K. DART: Dynamic Animation and Robotics Toolkit. *J. Open Source Softw.* **2018**, *3*, 500. [CrossRef]
49. Paul, S.; Chatzilygeroudis, K.; Ciosek, K.; Mouret, J.-B.; Osborne, M.A.; Whiteson, S. Alternating Optimisation and Quadrature for Robust Control. In Proceedings of the AAAI Conference on ArtificialIntelligence., New Orleans, LA, USA, 2–7 February 2018.
50. Yu, W.; Tan, J.; Liu, C.K.; Turk, G. Preparing for the Unknown: Learning a Universal Policy with Online System Identification. In Proceedings of the 13th Robotics: Science and Systems, Cambridge, MA, USA, 12–16 July 2017.
51. Chatzilygeroudis, K.; Mouret, J.-B. Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018.
52. Yu, W.; Liu, C.K.; Turk, G. Multi-task Learning with Gradient Guided Policy Specialization. *CoRR Abs* **2017**, *5*, 257–270. [CrossRef]
53. Chatzilygeroudis, K.; Vassiliades, V.; Mouret, J.-B. Reset-free Trial-and-Error Learning for Robot Damage Recovery. *Robot. Auton. Syst.* **2016**, *100*, 14. [CrossRef]
54. Kumar, V.C.V.; Ha, S.; Liu, C.K. Learning a Unified Control Policy for Safe Falling. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017.
55. Clegg, A.; Yu, W.; Erickson, Z.; Tan, J.; Liu, C.K.; Turk, G. Learning to Navigate Cloth using Haptics. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017.
56. Clegg, A.; Yu, W.; Tan, J.; Kemp, C.C.; Turk, G.; Liu, C.K. Learning Human Behaviors for Robot-Assisted Dressing. *arXiv* **2017**, arXiv:1709.07033. [CrossRef]
57. Todorov, E.; Erez, T.; Tassa, Y. MuJoCo: A physics engine for model-based control. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 7–12 October 2012.
58. Nachum, O.; Norouzi, M.; Xu, K.; Schuurmans, D. Trust-PCL: An Off-Policy Trust Region Method for Continuous Control. In Proceedings of the International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 24–28 September 2018.
59. Wu, Y.; Mansimov, E.; Liao, S.; Grosse, R.; Ba, J. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS), Long Beach, CA, USA, 4–9 December 2017.
60. Hwangbo, J.; Lee, J.; Hutter, M. Per-Contact Iteration Method for Solving Contact Dynamics. *IEEE Robot. Autom. Lett.* **2018**, *3*, 895–902. [CrossRef]

61. Carius, J.; Ranftl, R.; Farshidian, F.; Hutter, M. Constrained stochastic optimal control with learned importance sampling: A path integral approach. *Int. J. Rob. Res.* **2022**, *41*, 189–209. [CrossRef] [PubMed]
62. Tsounis, V.; Alge, M.; Lee, J.; Farshidian, F.; Hutter, M. DeepGait: Planning and Control of Quadrupedal Gaits using Deep Reinforcement Learning. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020.
63. Lee, J.; Hwangbo, J.; Wellhausen, L.; Koltun, V.; Hutter, M. Learning Quadrupedal Locomotion over Challenging Terrain. *Sci Robot.* **2020**, *5*, eabc5986. [CrossRef]
64. Lee, J.; Hwangbo, J.; Hutter, M. Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning. *arXiv* **2019**, arXiv:1901.07517.
65. Shi, F.; Homberger, T.; Lee, J.; Miki, T.; Zhao, M.; Farshidian, F.; Okada, K.; Inaba, M.; Hutter, M. Circus ANYmal: A Quadruped Learning Dexterous Manipulation with Its Limbs. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021.
66. Kang, D.; Hwangho, J. Physics engine benchmark for robotics applications RaiSim vs. Bullet vs. ODE vs. MuJoCo vs. DartSim. *RaiSim Platf.* **2018**. Available online: https://leggedrobotics.github.io/SimBenchmark/ (accessed on 14 March 2023).
67. Wang, Z. *Learning to Land on Flexible Structures*; KTH: Stockholm, Sweden, 2022.
68. Corporation, N. Nvidia Isaac Sim. Available online: https://developer.nvidia.comisaac-sim (accessed on 14 March 2023).
69. Audonnet, F.P.; Hamilton, A.; Aragon-Camarasa, G. A Systematic Comparison of Simulation Software for Robotic Arm Manipulation using ROS2. In Proceedings of the 22nd International Conference on Control, Automation and Systems (ICCAS), BEXCO, Busan, Republic of Korea, 27–30 November 2022.
70. Monteiro, F.F.; Vieira-e-Silva, A.L.B.; Teixeira, J.M.X.N.; Teichrieb, V. Simulating real robots in virtual environments using NVIDIA's Isaac SDK. In Proceedings of the XXI Symposium on Virtual and Augmented Reality, Natal, Brazil, 28–31 October 2019.
71. Makoviychuk, V.; Wawrzyniak, L.; Guo, Y.; Lu, M.; Storey, K.; Macklin, M.; Hoeller, D.; Rudin, N.; Allshire, A.; Handa, A.; et al. Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning. *arXiv* **2021**, arXiv:2108.10470. [CrossRef]
72. Rojas, M.; Hermosilla, G.; Yunge, D.; Farias, G. An Easy to Use Deep Reinforcement Learning Library for AI Mobile Robots in Isaac Sim. *Appl. Sci.* **2022**, *12*, 8429. [CrossRef]
73. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In Proceedings of the Field and Service Robotics conference (FSR), Zurich, Switzerland, 12–15 September 2017.
74. Shin, S.-Y.; Kang, Y.-W.; Kim, Y.-G. Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot. *Appl. Sci.* **2019**, *9*, 5571. [CrossRef]
75. Park, J.-H.; Farkhodov, K.; Lee, S.-H.; Kwon, K.-R. Deep Reinforcement Learning-Based DQN Agent Algorithm for Visual Object Tracking in a Virtual Environmental Simulation. *Appl. Sci.* **2022**, *12*, 3220. [CrossRef]
76. Wu, T.-C.; Tseng, S.-Y.; Lai, C.-F.; Ho, C.-Y.; Lai, Y.-H. Navigating Assistance System for Quadcopter with Deep Reinforcement Learning. In Proceedings of the 1st International Cognitive Cities Conference (IC3), Okinawa, Japan, 7–9 August 2018.
77. Anwar, A.; Raychowdhury, A. Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes using Transfer Learning. *arXiv* **2019**, arXiv:1910.05547. [CrossRef]
78. Yoon, I.; Anwar, M.A.; Joshi, R.V.; Rakshit, T.; Raychowdhury, A. Hierarchical Memory System With STT-MRAM and SRAM to Support Transfer and Real-Time Reinforcement Learning in Autonomous Drones. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2019**, *9*, 485–497. [CrossRef]
79. Tanenbaum, A.S. *Computer Networks Tanenbaum*; Pearson Education: Upper Saddle River, NJ, USA, 2003.
80. Peterson, L.L.; Davie, B.S. *ComputerNetworks: A Systems Approach*; Morgan Kaufmann: Burlington, NJ, USA, 2012.
81. Grooten, B.; Wemmenhove, J.; Poot, M.; Portegies, J. Is Vanilla Policy Gradient Overlooked? Analyzing Deep Reinforcement Learning for Hanabi. In Proceedings of the ALA(Adaptive and Learning Agents Workshop at AAMAS), Auckland, New Zealand, 9–10 May 2022.
82. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS-99), Cambridge, MA, USA, 29 November–4 December 1999; pp. 1057–1063.
83. Two-Way Communication between Python 3 and Unity (C#)-Y. T. Elashry. Available online: https://github.com/Siliconifier/Python-Unity-Socket-Communication (accessed on 14 March 2023).
84. Unity Asset Store. Available online: https://assetstore.unity.com/packages/3d/vehicles/air/simple-drone-190684#description (accessed on 14 March 2023).
85. Sketchfab Platform for Ready 3D Models. Available online: https://sketchfab.com/3d-models/airport-c26922efb90c44988522d4638ad5d217 (accessed on 14 March 2023).
86. Schulman, J. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*; EECS Department, University of California: Berkeley, CA, USA, 2016.
87. Abbass, M.A.B.; Hamdy, M. A Generic Pipeline for Machine Learning Users in Energy and Buildings Domain. *Energies* **2021**, *14*, 5410. [CrossRef]
88. Abbass, M.A.B.; Sadek, H.; Hamdy, M. Buildings Energy Prediction Using Artificial Neural Networks. *Eng. Res. J. EJR* **2021**, *171*, 12.

89. A Video Link for Actor-Critic Algorithm to Control Drone. Available online: https://youtu.be/OyNK6QSuMuU (accessed on 14 March 2023).
90. A Video Link for Vanilla Policy Gradient Algorithm to Control Drone. Available online: https://youtu.be/r-DKqIC1bGI (accessed on 14 March 2023).
91. A Video Link for Overall Python-Unity Integrated Platform in Runtime. Available online: https://youtu.be/ZQzC05qr_q0 (accessed on 14 March 2023).