Autonomous Unmanned Aerial Vehicle Navigation using Reinforcement Learning: A Systematic Review

Fadi AlMahamid, Katarina Grolinger*

Department of Electrical and Computer Engineering, Western University, London, Ontario, Canada

ARTICLE INFO

Keywords:
Reinforcement Learning
Autonomous UAV Navigation
UAV
Systematic Review

ABSTRACT

There is an increasing demand for using Unmanned Aerial Vehicle (UAV), known as drones, in different applications such as packages delivery, traffic monitoring, search and rescue operations, and military combat engagements. In all of these applications, the UAV is used to navigate the environment autonomously - without human interaction, perform specific tasks and avoid obstacles. Autonomous UAV navigation is commonly accomplished using Reinforcement Learning (RL), where agents act as experts in a domain to navigate the environment while avoiding obstacles. Understanding the navigation environment and algorithmic limitations plays an essential role in choosing the appropriate RL algorithm to solve the navigation problem effectively. Consequently, this study first identifies the main UAV navigation tasks and discusses navigation frameworks and simulation software. Next, RL algorithms are classified and discussed based on the environment, algorithm characteristics, abilities, and applications in different UAV navigation problems, which will help the practitioners and researchers select the appropriate RL algorithms for their UAV navigation use cases. Moreover, identified gaps and opportunities will drive UAV navigation research.

Symbol	Description
$s \in S$	State s belongs to all possible states S
$a \in A$	Action a belongs to the set of all possible Actions A
$r \in R$	Reward r belongs to the set of all generated Rewards R
γ	Discounted factor γ used to reduce the impact the future rewards, where $0 < \gamma < 1$
G_t	The Expected Summation of the Discounted Rewards; $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
$P(s', r\|s, a)$	The probability of the transition to state s' with reward r from taking action a in state s at time t
τ	A trajectory τ consists of a sequence of actions and states pairs, where the actions influence the states, also called an episode. Each trajectory has a start state and ends in a final state that terminates the trajectory
$Q(s, a)$	Action-value function measures the expected return of the state and action pairs (s, a); $Q^w(s, a)$ is the action-value function parameterized by w
$V(s)$	State-value function is similar to $Q(s, a)$ except it measures how good to be in a state s; $V^w(s)$ is a State-value function parameterized by w
$A(s, a)$	Advantage-Value function $A(s, a)$ captures how good is an action compared to other actions at a given state; $A(s, a) = Q(s, a) - V(s)$
$\pi(a\|s)$	Stochastic Policy π is a function that maps the probability of selecting an action a from the state s. It describes agent behavior
$\mu(s)$	Deterministic Policy μ is similar to Stochastic Policy π, except μ symbol is used to distinguish it from Stochastic Policy π
$Q_\pi(s, a)$	Action-value function $Q(s, a)$, when following a policy π
$V_\pi(s)$	State-value function $V(s)$, when following a policy π
ρ	State visitation probability
$\sim$	Sampled from. For example, $s \sim \rho$ means s sampled from state visitation probability ρ
$\eta(\pi)$	The expected discounted reward following a policy π, similar to G_t

Table 1
Mathematical Notations

1. Introduction

Autonomous Systems (AS) are defined as systems that can perform desired tasks without human interference. For example, robots performing tasks without human involvement, self-driving cars, and delivery drones. AS are invading different domains to make operations more efficient and reduce the cost and risk incurred from the human factor.

An Unmanned Aerial Vehicle (UAV) is an aircraft without a human pilot, mainly known as a drone. Autonomous UAVs have been receiving an increasing interest due to their diverse applications, such as delivering packages to customers, responding to traffic collisions to attain injured with medical needs, tracking military targets, assisting with search and rescue operations, and many other applications.

*Corresponding author.
falmaham@uwo.ca (F. AlMahamid); kgrolinger@uwo.ca (K. Grolinger)
ORCID(s): 0000-0002-6907-7626 (F. AlMahamid); 0000-0003-0062-8212 (K. Grolinger)

Typically, UAVs are equipped with cameras, among other sensors, that collect information from the surrounding environment, enabling UAVs to navigate that environment autonomously. UAV navigation training is typically conducted in a virtual 3D environment because UAVs have limited computation resources and power supply, and replacing UAV parts due to crashes can be expensive.

Different Reinforcement Learning (RL) algorithms are used to train UAVs to navigate the environment autonomously. RL can solve various problems where the agent acts as a human expert in the domain. The agent interacts with the environment by processing the environment's state, responding with an action, and receiving a reward. UAV cameras and sensors capture information from the environment for state representation. The agent processes the captured state and outputs an action that determines the UAV movement's direction or controls the propellers' thrust, as illustrated in Figure 1.

The research community provided a review of different UAV navigation problems, such as Visual UAV navigation [1, 2], UAV Flocking [3] and Path Planning [4]. Nevertheless, to the best of the authors' knowledge there is no survey related to applications of RL in UAV navigation. Hence, this paper aims to provide a comprehensive and systematic review on the application of various RL algorithms to different autonomous UAV navigation problems. This survey has the following contributions:

- Help the practitioners and researchers to select the right algorithm to solve the problem on hand based on the application area and environment type.
- Explain primary principles and characteristics of various RL algorithms, identify relationships among them, and classify them according to the environment type.
- Discuss and classify different RL UAV navigation frameworks according to the problem domain.

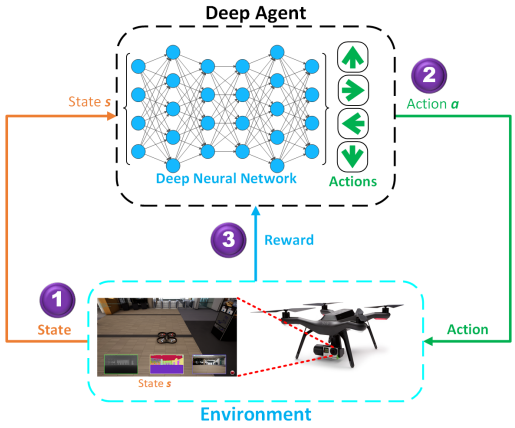

Figure 1: UAV training using deep reinforcement agent

- Recognize the various techniques used to solve different UAV autonomous navigation problems and the different simulation tools used to perform UAV navigation tasks.

The remainder of the paper is organized as follows: Section 2 presents the systematic review process, Section 3 introduces RL, Section 4 provides a comprehensive review of the application of various RL algorithms and techniques in autonomous UAV navigation, Section 5 discusses the UAV Navigation Frameworks and simulation software, Section 6 classifies RL algorithm and discusses the most prominent algorithms, and Section 7 identifies challenges and research opportunities. Finally, Section 8 concludes the paper.

2. Review Process

This section described the inclusion criteria, paper identification process, and threats to validity.

2.1. Inclusion Criteria and Identification of Papers

The study's main objective is to analyze the application of Reinforcement Learning in UAV navigation and provide insights into RL algorithms. Therefore, the survey considered all papers in the past five years (2016-2021) written in the English language that include the following terms combined, alongside with their variations: *Reinforcement Learning*, *Navigation*, and *UAV*.

In contrast, RL algorithms are listed based on the authors' domain knowledge of the most prominent algorithms and by going through the related work of the identified algorithms with no restriction to the publication time to include a large number of algorithms.

The identification process of the papers went through the following stages:

- First stage: The authors identified all the papers that strictly apply RL to UAV Navigation.
- Second stage: The authors listed all RL algorithms based on authors' knowledge of the most prominent algorithms, the references of recognized algorithms, then identified the corresponding paper of each algorithm.
- Third stage: the authors identified how RL is used to solve different UAV navigation problems, classified the work, and then recognized more related papers using exiting work references.

IEEE Xplore and Scopus were the primary sources of papers' identification between 2016 and 2021. The search query was applied using different terminologies that are used to describe the UAV alternatively, such as *UNMANNED AERIAL VEHICLE*, *DRONE*, *QUADCOPTER*, or *QUADROTOR*, and these terms are cross-checked with *REINFORCEMENT LEARNING*, and *NAVIGATION*, which resulted in a total of 104 papers. After removing 15 duplicate papers and 9 unrelated papers, the count became 81.

The authors identified another 69 papers that mainly describe the RL algorithms based on the authors' experience and the references list of the recognized work, using Google Scholar as the primary search engine. While RL for UAV navigation studies were restricted to five years, all RL algorithms are included as many are still extensively used regardless of their age. The search was completed in November 2021, with a total of 150 papers after all exclusions.

2.2. Threats to Validity

Despite the authors' effort to include all relevant papers, the study might be subject to the following main threats to the validity:

- Location bias: The search for papers was performed using two primary digital libraries (databases), IEEE Xplore and Scopus, which might limit the retrieved papers based on the published journals, conferences, and workshops in the database.
- Language bias: Only papers published in English are included.
- Time Bias: The search query is only limited to retrieving papers between 2016 and 2021, which results in excluding relevant papers published before 2016.
- Knowledge reporting bias: The research papers of RL algorithms are identified using authors' knowledge of variant algorithms and the related work in the recognized algorithms. It is hard to pinpoint all algorithms utilizing a search query, which could result in missing some RL algorithms.

3. Reinforcement Learning

The RL agent learns from taking actions in the environment, which causes a change in the environment's current state and generates a reward based on the action taken as expressed in the Markov Decision Process (MDP). We define the probability of the transition to state $\mathbf{s'}$ with reward $\boldsymbol{r}$ from taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$ at time $\boldsymbol{t}$, for all $s' \in S,\ s \in S,\ r \in R,\ a \in A(s)$, as:

$$P(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \quad (1)$$

The agent receives rewards for performing actions and uses them to measure the action's success or failure. The Reward R can be expressed in different forms, as a function of the action $R(a)$, or as a function of action-state pairs $R(a, s)$.

The agent's objective is to maximize the expected summation of the discounted rewards, which drives the agent to take the selected actions. The reward is granted by adding all the rewards generated from executing an episode. The *episode* (trajectory) represents a finite number of actions and ends when the agent achieves a final state, for example, when a collision occurs in a simulated navigation environment. However, in some cases, the actions can be continuous and cannot be broken into episodes. The discounted reward, as shown in equation 2 uses a multiplier $\boldsymbol{\gamma}$ to the power $\boldsymbol{k}$, where $\boldsymbol{\gamma \in [0, 1]}$. The value of k increases by one at each time step to emphasize the current reward and to reduce the impact of the future rewards, hence the term discounted reward.

$$G_t = E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right] \quad (2)$$

Emphasizing the current action's immediate reward and reducing the impact of future actions' rewards help the expected summation of discounted rewards to converge.

The following subsections introduce important reinforcement learning concepts.

3.1. Policy and Value Function

The agent's behavior is defined by following a policy $\boldsymbol{\pi}$, where the policy $\boldsymbol{\pi}$ defines the probability of taking action $\boldsymbol{a}$, given a state $\boldsymbol{s}$, which is denoted as $\boldsymbol{\pi(a|s)}$. Once the agent takes an action, the agent uses a value function to evaluate the action. The agent either uses: 1) a *state-value function* to estimate how good it is for the agent to be in state $\boldsymbol{s}$, or 2) a *action-value function* to measure how good it is for the agent to perform an action $\boldsymbol{a}$ in a given state $\boldsymbol{s}$. The action-value function is defined in terms of the expected summation of the discounted rewards and represents the target Q-value:

$$Q_\pi(s, a) = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \quad (3)$$

The agent performs the action with the highest Q-value, which might not be the optimal Q-value. Finding the optimal Q-value requires selecting the best actions that maximize the expected summation of discounted rewards under the optimal policy $\boldsymbol{\pi}$. The optimal Q-value $\boldsymbol{Q_*(s, a)}$ as described in equation 4 must satisfy the *Bellman optimality equation*, which is equal to the expected reward $\boldsymbol{R_{t+1}}$, plus the maximum expected discounted return that can be achieved for any possible next state-action pairs $\boldsymbol{(s', a')}$.

$$Q_*(s, a) = \max_{\pi} Q(s, a) \quad (4)$$

$$Q_*(s, a) = E\left[R_{t+1} + \gamma \max_{a'} Q_*(s', a')\right] \quad (5)$$

This optimal Q-value $\boldsymbol{Q_*(s, a)}$ is used to train the neural network. The Q-value $\boldsymbol{Q(s, a)}$ predicted by the network is subtracted from the optimal Q-value $\boldsymbol{Q_*(s, a)}$ estimated using the Bellman equation and backpropagated through the network. The loss function is defined as follows:

$$\overbrace{E\left[R_{t+1} + \gamma \max_{a'} Q_*(s', a')\right]}^{Target} - \overbrace{E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]}^{Predicted} \quad (6)$$

3.2. Exploration vs Exploitation

Exploration vs. Exploitation may be demonstrated using the multi-armed bandit dilemma, which accurately portrays the behavior of a person experiencing their first slot machine experience. The money (reward) player receives early in the game is unrelated to any previously selected choices, and as the player develops a comprehension of the reward, he/she begins selecting choices that contribute to earning a greater reward. The choices made randomly by the player to acquire knowledge might be defined as the player *Exploring* the environment. In contrast, the player's *Exploiting* the environment is described as the options selected based on his/her experience.

The RL agent needs to find the right balance between exploration and exploitation to maximize the expected return of rewards. Constantly exploiting the environment and selecting the action with the highest reward does not guarantee that the agent performs the optimal action because the agent may miss out on a higher reward provided by future actions taking alternative sets of actions in the future. Finding the ratio between exploration and exploitation can be defined through different strategies such as ϵ-greedy strategy, Upper Confidence Bound (UBC), and Gradient Bandits.

3.3. Experience Replay

In RL, an experience $\boldsymbol{e}$ can be described as the knowledge produced from the agent performing an action $\boldsymbol{a}$ in a state $\boldsymbol{s}$ causing a new state $\boldsymbol{s'}$ and a generated reward $\boldsymbol{r}$. The experience can be expressed as a tuple $\boldsymbol{e(s, a, s', r)}$. Lin [5] proposed a technique called *Experience Replay*, where experiences are stored in a replay memory (buffer) $\boldsymbol{D}$ and used to train the agent. Since experiences are stored in the memory, and some experiences might be of a high importance, they can repeatedly be reused to train the agent, which improves convergence.

Although experience replay should help the agent theoretically learn from previous important experiences, it entails sampling experiences uniformly from the replay memory $\boldsymbol{D}$ regardless of their significance. Schaul *et al.* [6] suggested the use of *Prioritized Experience Replay*, which aims to prioritize experiences using Temporal Difference error (TD-error) and replay more frequently experiences that have lower TD-error.

3.4. On-Policy vs Off-Policy

In order to interact with the environment, the RL agent attempts to learn two policies: the first one is referred to as the target policy $\boldsymbol{\theta(a|s)}$, which the agent learns through the value function, and the second one is referred to as the behavior policy $\boldsymbol{\beta(a|s)}$, which the agent uses for action selection when interacting with the environment.

When the training sample is collected according to the target policy and the expected return is generated for the same policy, the algorithm is referred to as *on-policy algorithm*. On the other hand, in *off-policy algorithms*, the training sample follows a behavior policy, while the expected reward is generated using the target policy [7]. Off-policy algorithms do not require full trajectories (episodes) for the training sample, and they can reuse past trajectories.

3.5. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) uses deep agents to learn the optimal policy where it combines artificial Neural Networks (NN) with Reinforcement Learning (RL). The NN type used in DRL varies from one application to another depending on the problem being solved, inputs type (state), and the number of inputs passed to the NN. For example, the RL framework can be integrated with Convolutional Neural Network (CNN) to process images representing the environment's state or combined with Recurrent Neural Network (RNN) to process inputs over different time steps.

The architecture of the deep agent can be simple or complex based on the problem at hand, where a complex architecture combines multiple NN. But what all deep agents have in common is that they receive the state as an input, then they output the optimal action and maximize the discounted return of rewards.

The application of Deep NN to the RL framework enabled the research community to solve more complex problems in autonomous systems that were impossible to solve before, such as drone navigation and avoiding obstacles using images received from the drone's monocular camera.

4. Autonomous UAV Navigation using DRL

Different DRL algorithms and techniques were used to solve various problems in autonomous UAV navigation, such as UAV control, obstacle avoidance, path planning, and flocking. The DRL agent acted as an expert in all of these problems, selecting the best action that maximizes the reward to achieve the desired objective. The input and the output of the DRL algorithm are generally determined based on the desired objective and the implemented technique. The following subsections summarize the use of DRL in the main UAV navigation tasks.

4.1. UAV Control

RL is used to control the movement of the UAV in the environment by applying changes to the flight mechanics of the UAV, which varies based on the UAV type. In general UAVs can be classified based on the flight mechanics into 1) Multirotor, 2) Fixed-Wing, and 3) single-rotor, and 4) fixed-wing hybrid Vertical Take-Off and Landing (VTOL) [8].

Multirotor, also known as multicopter or drone, uses more than two rotors to control the flight mechanics by applying different amounts of thrust to the rotors causing changes in principal axes leading to four UAV movements 1) pitch, 2) roll, 3) yaw, and 4) throttle as explained in Figure 2. Similarly, single-rotor and fixed-wing hybrid VTOL apply changes to different rotors to generate the desired movement, except they both use tilt-rotor(s) and wings in fixed-wing hybrid VTOL. On the other hand, fixed-wing can only achieve three actions pitch, roll, and yaw, where they take off by generating enough speed that causes the air-dynamics to lift-up the UAV.

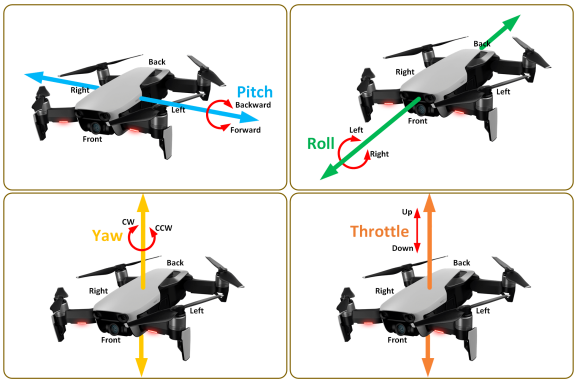

Figure 2: Multirotor Flight Mechanics

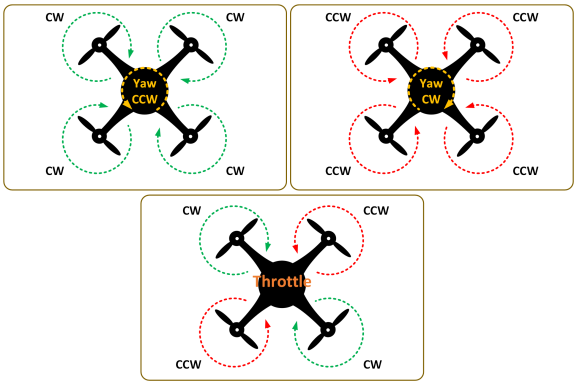

Figure 3: Yaw vs Throttle Mechanics

Quad-rotors has four propellers: two diagonal propellers rotate clockwise and the other two propellers rotate counter-clockwise causing the throttle action. When the propellers generate a thrust more significant than the UAV weight they cause elevation, and when the thrust power equals the UAV weight, the UAV stops elevation and starts hovering in place. In contrast, if all propellers rotate in the same direction, they cause a yaw action in the opposite direction, as shown in Figure 3.

The steps described in Figure 4 depicts the RL process used to control the UAV, which depends on the used RL algorithm, but the most important takeaway is that RL uses the UAV state to produce actions. These actions are responsible for moving the UAV in the environment and can be either direct changes in the value of pitch, roll, yaw, and throttle values or indirect changes that require transformation to commands understood by the UAV.

4.2. Obstacle Avoidance

Avoiding obstacles is an essential task required by the UAV to navigate any environment, which can be achieved by estimating the distance to the objects in the environment using different devices such as front-facing cameras or distance sensors. The output generated by these different devices provides input to the RL algorithm and plays a significant role in the NN architecture.

Yuncheng *et al.* [1] described different front-facing cameras such as monocular cameras, stereo cameras, and RGB-D cameras that a UAV can use. Each camera type produces a different image type used as raw input to the RL agent. However, regardless of the camera type, these images can be preprocessed using computer vision to produce specific image types as described below:

- RGB Images: are renowned colored images where each pixel is represented in three values (Red, Green, Blue) ranging between $(0, 255)$.
- Depth-Map Images: contains information related to the distance of the objects from the Field Of View (FOV).
- Event-Based Images: are special images that output the changes in brightness intensity instead of standard images. Event-based images are produced by an event camera, also known as Dynamic Vision Sensor (DVS).

RGB images lack depth information, and, therefore, the agent cannot estimate how far or close the UAV is to the object leading to unexpected flying behavior. On the other hand, depth information is essential for building a successful reward function that penalizes moving closer to the objects. Some techniques used RGB images and depth-map simultaneously as input to the agent to provide more information about the environment. In contrast, event-based images data are represented as one-dimensional sequences of events over time, which is used to capture quickly changing information in the scene [9].

Similar to cameras, distance sensors have different types, such as LiDAR, RADAR, and acoustic sensors: they estimate the distance of the surrounding objects to the UAV but require less storage size than 2D images since they do not use RGB channels.

The output generated by these devices reflects the different states that the UAV has over time, used as an input to the RL agent to make actions causing the UAV to move in different directions to avoid obstacles. The NN architecture of the RL agent is based on: 1) input type, 2) the number of inputs, and 3) the used algorithm. For example, processing RGB images or depth-map images using the DQN algorithm requires Convolutional Neural Network (CNN) followed by fully-connect layers since CNN is known for its power in processing images. In contrast, processing event-based images is performed using Spiking Neural Networks (SNN), which is designed to handle spatio-temporal data and identify spatio-temporal patterns [9].

4.3. Path Planning

Autonomous UAVs must have a well-defined objective before executing a flying mission. Typically, the goal is to fly from a start to a destination point, such as in delivery drones. But, the goal can also be more sophisticated, such as performing surveillance by hovering over a geographical

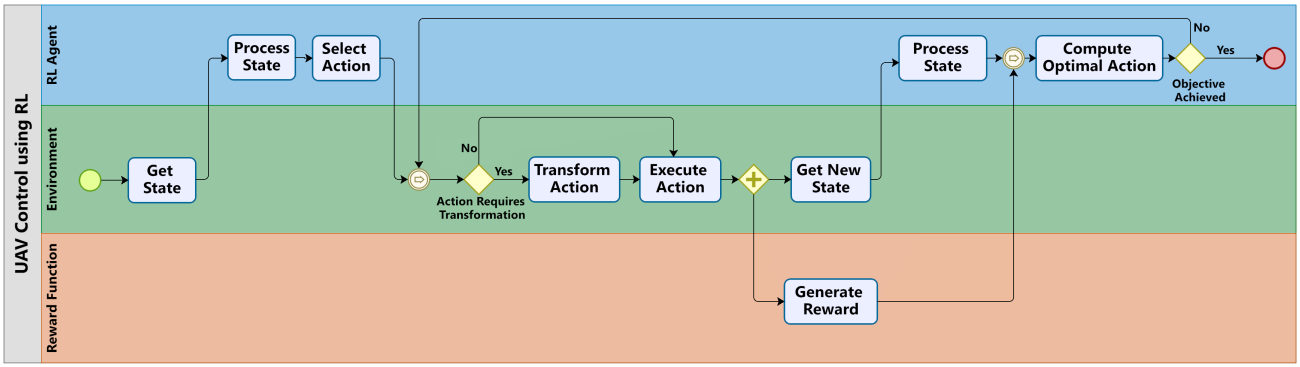

Figure 4: UAV Control using RL

area or participating in search and rescue operations to find a missing person.

Autonomous UAV navigation requires path planning to find the best UAV path to achieve the flying objective while avoiding obstacles. The optimal path does not always mean the shortest path or a straight line between two points; instead, the UAV aims to find a safe path while considering UAV's limited power and flying mission.

Path planning can be divided into two main types:

- Global Path Planning: concerned with planning the path from the start point to destination point in attempt to select the optimal path.
- Local Path Planning: concerned with planning the local optimal waypoints in an attempt to avoid static and dynamic obstacles while considering the final destination.

Path planning can be solved using different techniques and algorithms; in this work, we focus on RL techniques used to solve global and local path planning, where the RL agent receives information from the environment and outputs the optimal waypoints according to the reward function. RL techniques can be classified according to the usage of the environment's local information 1) map-based navigation and 2) mapless navigation.

4.3.1. Map-Based Navigation

A UAV that adopts map-based navigation uses a representation of the environment either in 3D or 2D format. The representation might include one or more of the following about the environment: 1) the different terrains, 2) fixed-obstacles locations, and 3) charging/ground stations.

Some maps oversimplify the environment representation: the map is divided into a grid with equally-sized smaller cells that store information about the environment [10–12]. Others oversimplify the environment's structure by simplifying objects representation or by using 1D/2D to represent the environment [13–23] . The UAV has to plan a safe and optimal path over the cells to avoid cells containing obstacles until it reaches its destination and has to plan its stopover at the charging stations based on the battery level and path length.

In a more realistic scenario, the UAV calculates a route using the map information and the GPS signal to track the UAV's current location, starting point, and destination point. The RL agent evaluates the change in the distance and the angle between the UAV's current GPS location and target GPS location, and penalizes the reward if the difference increases or if the path is unsafe depending on the reward function (objective).

4.3.2. Mapless Navigation

Mapless navigation does not rely on maps; instead, it applies computer vision techniques to extract features from the environment and learn the different patterns to reach the destination, which requires computation resources that might be overwhelming for some UAVs.

Localization information of the UAV obtained by different means such as Global Positioning System (GPS) or Inertial Measurement Unit (IMU) is used in mapless navigation to plan the optimal path. DRL agent receives the scene image, the destination target, and the localization information as input and outputs the change in the UAV movements.

For example, Benchun *et al.* [24] calculated and tracked the angle between the UAV and destination point, then encoded it with the depth image extracted from the scene and used both as a state representation for the DRL agent. Although localization information seems essential to plan the path, some techniques achieved navigation with high speed using monocular visual reactive navigation system without a GPS [25].

4.4. Flocking

Although UAVs are known for performing individual tasks, they can flock to perform tasks efficiently and quickly, which requires maintaining flight formation. UAV flocking has many applications, such as search and rescue operations to cover a wide geographical area.

UAV flocking is considered a more sophisticated task than a single UAV flying mission because UAVs need to orchestrate their flight to maintain flight formation while

performing other tasks such as UAV control and obstacle avoidance. Flocking can be executed using different topologies:

- Flock Centering: maintaining flight formation as suggested by Reynolds [26] involves three concepts: 1) flock centering, 2) avoiding obstacles, and 3) velocity matching. This topology was applied in several research papers [27–31].
- Leader-Follower Flocking: the flock leader has its mission of reaching destination, while the followers (other UAVs) flock with the leader with a mission of maintaining distance and relative position to the leader [32, 33].
- Neighbors Flocking: close neighbors coordinate with each other, where each UAV communicates with two or more nearest neighbors to maintain flight formation by maintaining relative distance and angle to the neighbors [34–36].

Maintaining flight formation using RL requires communication between UAVs to learn the best policy to maintain the formation while avoiding obstacles. These RL systems can be trained using a single agent or multi-agents in centralized or distributed settings.

4.4.1. Centralized Training

A centralized RL agent trains a shared flocking policy to maintain the flock formation using the experience collected from all UAVs, while each UAV acts individually according to its local environment information such as obstacles. The reward function of the centralized agent can be customized to serve the flocking topology, such as flock centering or leader-follower flocking.

Yan *et al.* [16] used Proximal Policy Optimization (PPO) algorithm to train a centralized shared flocking control policy, where each UAV flocks as close as possible to the center and decentralized execution for obstacle avoidance according to each UAV local environment information. Similarly, Hung and Givigi [33] trained a leader UAV to reach a destination while avoiding obstacles and trained a shared policy for followers to flock with the leader considering the relative dynamics between the leader and the followers.

Zhao *et al.* [37] used a Multi-Agent Reinforcement Learning (MARL) to train a centralized flock control policy shared by all UAVs with decentralized execution. MARL received position, speed, and flight path angle from all UAVs at each time step and tried to find the optimal flocking control policy.

The centralized training would not produce a good generalization in neighbors flocking topology since the learned policy for one neighbor is different from other neighbors' policies due to the differences in neighbors' dynamics.

4.4.2. Distributed Training

UAV flocking can be trained using a distributed (decentralized) approach, where each UAV has its designated RL agent responsible for finding the optimal flock policy for the UAV. The reward function is defined to maintain distance and flying direction with other UAVs and can be customized to include further information depending on the objective.

Flight information such as location and heading angle should be communicated to other UAVs since the RL agents are distributed, and the state representation must include information of other UAVs. Any UAV that fails to receive the information from other UAVs will cause the UAV to be isolated from the flock.

Liu *et al.* [15] proposed a decentralized DRL framework to control each UAV in a distributed setting to maximize average coverage score, geographical fairness, and minimize UAVs' energy consumption.

Framework Objective	Papers
Energy-aware UAV Navigation	[38, 39]
Path Planning	[20, 40–45]
Flocking	[46, 47]
Vision-Based Frameworks	[11, 48–50]
Transfer Learning	[51]

Table 2
UAV Navigation Frameworks

5. UAV Navigation Frameworks and Simulation Software

The following subsections discuss and classify the UAV navigation frameworks based on the intended framework objective and introduce different software used to execute the simulations.

5.1. UAV Navigation Frameworks

In general, a software framework is a conceptual structure analogous to a blueprint used to guide the comprehending construction of the software by defining different functions and their interrelationships. By definition, RL can be considered a framework by itself. Therefore, we considered only UAV navigation frameworks that add to traditional navigation using sensors or camera data for navigation. As a result, Table 2 classifies UAV frameworks based on the framework objective. The subsequent sections discuss the frameworks in more detail.

5.1.1. Energy-Aware UAV Navigation Frameworks

UAVs has limited flight time, hence operate mainly using batteries. Therefore, planning flight route and recharge stopover are crucial to reach destinations. Energy-aware UAV navigation frameworks aim to provide obstacles avoidance navigation while considering the UAV battery capacity.

Bouhamed *et al.* [38] developed a framework based on Deep Deterministic Policy Gradient (DDPG) algorithm to guide the UAV to a target position while communicating with ground stations, allowing the UAV to recharge its battery if it drops below a specific threshold. Similarly, Imanberdiyev *et al.* [39] monitor battery level, rotors' condition,

and sensor readings to plan the route and apply necessary route changes for required battery charging.

5.1.2. Path Planning Frameworks

Path planning is the process of determining the most efficient route that meets the flight objective, such as finding the shortest, fastest, or safest route. Different frameworks [20, 40] implemented a modular path planning scheme, where each module has a specialized function to achieve while exchanging data with other modules to train action selection policies and discover the optimal path.

Similarly, Li *et al.* [42] developed a four-layer framework in which each layer generates a set of objective and constraint functions. The functions are intended to serve the lower layer and consider the upper layer's objectives and constraints, with their primary goal generating trajectories.

Other frameworks suggested stage-based learning to choose actions from the desired stage depending on current environment encounters. For example, Camci and Kayacan [44] proposed learning a set of motion primitives offline, then using them online to design quick maneuvers to enable switching seamlessly between two modes: *near-hover motions*, which is responsible for generating motion plans allowing a stable completion of maneuvers and *swift maneuvers* to deal smoothly with abrupt inputs.

In a collaborative setting, Zhang *et al.* [43] suggested a coalition between Unmanned Ground Vehicle (UAG) and UAV complementing each other to reach the destination, where UAV cannot get to far locations alone due to limited battery power, and UGV cannot reach high altitude due to limited abilities.

5.1.3. Flocking Frameworks

UAV flocking frameworks have functionality beyond UAV navigation while maintaining flight formation. For example, Bouhamed *et al.* [46] presented a RL-based spatiotemporal scheduling system for autonomous UAVs. The system enables UAVs to autonomously arrange their schedules to cover the most significant number of pre-scheduled events physically and temporally spread throughout a specified geographical region and time horizon. On the other hand, Majd *et al.* [47] predicted the movement of drones and dynamic obstacles in the flying zone to generate efficient and safe routes.

5.1.4. Vision-Based Frameworks

Vision-Based Framework depends on UAV camera for navigation, where the images produced by the camera are used to draw on additional functionality for improved navigation. It is possible to employ frameworks that augment the agent's CNN architecture to fuse data from several sensors, use Long-Short Term Memory cells (LSTM) to maintain navigation choices, use RNN to capture the UAV states over different time steps, or pre-process images to provide more information about the environment [48, 48, 49].

5.1.5. Transfer Learning Frameworks

UAVs are trained on target environments before executing the flight mission; the training is carried either in a virtual or real-world environment. The UAV requires retraining when introduced to new environments or moving from virtual training as the environments have different terrains and obstacle structures or textures. Besides, UAV training requires a long time and it is hardware resource intensive while actual UAVs have limited hardware resources. Therefore, when UAV is introduced to new environments, transfer learning frameworks reduce the training time by reusing the NN weights trained from the previous environment and retraining only parts of the agent's NN.

Yoon *et al.* [51] proposed algorithm-hardware co-design, where the UAV is trained in a virtual environment, and after the UAV is deployed to a real-world environment; the agent loads the weights stored in embedded Non-Volatile Memory (eNVM), and then evaluates new actions and only trains the last few layers of CNN whose weights are stored in the on-die SRAM (Static Random Access Memory).

5.2. Simulation Software

The research community used different evaluation methods for autonomous UAV navigation using RL. Simulation software is used widely over actual UAVs to execute the evaluation due to the cost of the hardware (drone) in addition to the cost of replacement parts required due to UAV crashes. Comparison between simulation software is not the intended purpose, rather than making the research community aware of the most commonly used tools for evaluation as illustrated in Figure 5. 3D UAV navigation simulation requires mainly three components as illustrated in Figure 6:

- **RL Agent:** represents the RL algorithm used with all computations required to generate the reward, process the states, and compute the optimal action. RL Agent

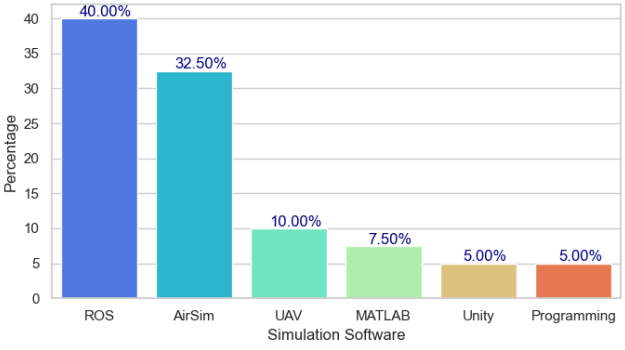

Figure 5: UAV Simulation Software Usage

Figure 6: UAV Simulation Software Components

interacts directly with the *UAV Flight simulator* to send/receive UAV actions/states.

- **UAV Flight Simulator:** responsible for simulating the UAV movements and interactions with the 3D environment, such as obtaining images from the UAV camera or reporting UAV crashes with different obstacles. Examples of UAV flight simulators are Robot Operating systems (ROS) [52] and Microsoft AirSim [53].
- **3D Graphics Engine:** provides a 3D graphics environment with the physics engine, which is responsible for simulating the gravity and dynamics similar to the real world. Examples of 3D graphics engines are Gazebo [54] and Unreal Engine [55].

Due to compatibility/support issues, ROS is used in conjunction with Gazebo, where AirSim uses Unreal Engine to run the simulations. However, the three components might not always be present, especially if the simulation software has internal modules or plugins that provide the required functionality, such as MATLAB.

6. Reinforcement Learning Algorithms Classification

While most reinforcement learning algorithms use deep neural networks, different algorithms are suited for different environment types. According to the number of the states and action types, RL algorithms can be classified as into three main categories: 1) a limited number of states and discrete actions, 2) an unlimited number of states and discrete actions, and 3) an unlimited number of states and continuous actions [56]. We extend this with sub-classes, analyze 51 RL algorithms, and examine their use in UAV navigation. Table 3 classifies all RL algorithms found in UAV Navigation studies and includes other prominent RL algorithms to show the intersection between RL and UAV navigation. Furthermore, this section discusses algorithms characteristics and highlights RL applications in different UAV navigation studies. Note that Table 3 includes many algorithms, but only the most prominent ones are discussed in the following subsections.

6.1. Limited States and Discrete Actions

The environments with discrete actions and limited states are relatively simple environments where the agent can select from pre-defined actions and be in pre-defined known states. For example, when an agent is playing a tic-tac-toe game, the nine boxes represent the states, and the agent can choose from two actions: X or O, and update the available states.

Q-Learning [57] algorithm is commonly used to solve problems in such environments. This algorithm finds the optimal policy in a Markov Decision Process (MDP) by maintaining a Q-Table table with all possible states and actions and iteratively updating the Q-values for each state-action pair using the Bellman equation until the Q-function converges to the optimal Q-value.

State–Action–Reward–State–Action (SARSA) [65] is another algorithm from this category: it is similar to Q-learning except it updates the current $\boldsymbol{Q(s,a)}$ value in a different way. In Q-learning, in order to update the current $\boldsymbol{Q(s,a)}$ value, we need to compute the next state-action $\boldsymbol{Q(s',a')}$ value, and since the next action is unknown, then Q-learning takes a greedy action to maximize the reward [104]. In contrast, when SARSA updates the current state-action $\boldsymbol{Q(s,a)}$ value, it performs the next action $\boldsymbol{a'}$ [104].

6.2. Unlimited States and Discrete Actions

In some environments, such as playing a complex game, the states can be limitless; however, the agent's choice is limited to a finite set of actions. In such environments, the agent mainly consists of a Deep Neural Network (DNN), usually a CNN, responsible for processing and extracting features from the state of the environment and outputting the available actions. Different algorithms can be used with this environment type, such as DQN, Deep SARSA, and their variants.

6.2.1. Deep Q-Networks Variations

Deep Q-Learning, also referred to as Deep Q-Networks (DQN) [66], is considered the main algorithm used in environments with unlimited states and discrete actions, and it inspires other algorithms used for a similar purpose. DQN usually combines convolutional and pooling layers, followed by fully connected layers that produce Q-values corresponding to the number of actions. Figure 7 [83] shows AlexNet CNN followed by two fully connected layers to produce Q-value. The current scene from the environment represents the environment's current state; once it is passed to the network, it produces Q-value representing the best action to take. The agent acts and then captures the changes in the environment's current state and the reward generated from the action.

A significant drawback of the DQN algorithm is that it overestimates the action-value (Q-value), where the agent tends to choose a non-optimal action because it has the highest Q-value [141].

Double DQN uses two networks to solve this overestimation problem in DQN. The first network, called the Policy Network, optimizes the Q-value, and the second network, the Target Network, is a replica of the policy network, and

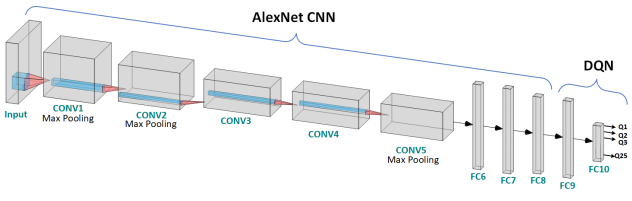

Figure 7: DQN using AlexNet CNN

State	Action	Algorithm	Class	On/Off Policy	Actor-Critic	Multi-Thread	Distributed	Multi-Agent	Usage
Limited	Discrete	Q-Learning [57]	Simple RL	-	No	No	No	No	[12, 19, 21, 22, 46, 58–64]
		SARSA [65]		-	No	No	No	No	-
Unlimited	Discrete	DQN [66]	DQN Variations	Off	No	No	No	No	[9, 17, 20, 23, 44, 45, 50, 63, 67–75, 75–80]
		Double DQN [81]		Off	No	No	No	No	[51, 63, 68, 73, 82–88]
		Dueling DQN [89]		Off	No	No	No	No	[68, 73]
		DRQN [90]		Off	No	No	No	No	[11, 49, 91, 92]
		DD-DQN [89]		Off	No	No	No	No	[68, 93]
		DD-DRQN		Off	No	No	No	No	-
		Noisy DQN [94]	Distributional DQN	Off	No	No	No	No	-
		C51-DQN [95]		Off	No	No	No	No	-
		QR-DQN [96]		Off	No	No	No	No	-
		IQN [97]		Off	No	No	No	No	-
		Rainbow DQN [98]		Off	No	No	No	No	-
		FQF [99]		Off	No	No	No	No	-
		R2D2 [100]	Distributed DQN	Off	No	No	Yes	No	-
		Ape-X DQN [101]		Off	No	No	Yes	No	-
		NGU [102]		Off	No	No	Yes	No	-
		Agent57 [103]		Off	No	No	Yes	No	-
		Deep SARSA [104]	Deep SARSA Variations	On	No	No	No	No	-
		Double SARSA		On	No	No	No	No	-
		Dueling SARSA		On	No	No	No	No	-
		DR-SARSA		On	No	No	No	No	-
		DD-SARSA		On	No	No	No	No	-
		DD-DR-SARSA		On	No	No	No	No	-
	Continous	REINFORCE [105]	Policy Based	On	No	No	No	No	-
		TPRO [106]		On	No	No	No	No	[107]
		PPO [108]		On	No	No	No	No	[16, 43, 107, 109–114]
		PPG [115]		Off	No	No	No	No	-
		SVPG [116]		Off	No	No	No	No	-
		SLAC [117]	Actor-Critic	Off	Yes	No	No	No	-
		ACE [118]		Off	Yes	Yes	No	No	-
		DAC [119]		Off	Yes	No	No	No	-
		DPG [7]		Off	Yes	No	No	No	[120]
		RDPG [121]		Off	Yes	No	No	No	-
		DDPG [122]		Off	Yes	No	No	No	[13–15, 24, 34, 38, 41, 42, 48, 61, 107, 123–125]
		TD3 [126]		Off	Yes	No	No	No	[18]
		SAC [127]		Off	Yes	No	No	No	[13]
		Ape-X DPG [101]	Multi-Agent and Distributed Actor-Critic	Off	Yes	No	Yes	No	-
		D4PG [128]		Off	Yes	No	Yes	Yes	-
		A2C [129]		On	Yes	Yes	Yes	No	[28, 91]
		DPPO [130]		On	No	No	Yes	Yes	-
		A3C [129]		On	Yes	Yes	No	Yes	[14]
		PAAC [131]		On	Yes	Yes	No	No	-
		ACER [132]		Off	Yes	Yes	No	No	-
		Reactor [133]		Off	Yes	Yes	No	No	-
		ACKTR [134]		On	Yes	Yes	No	No	-
		MADDPG [135]		Off	Yes	No	No	Yes	-
		MATD3 [136]		Off	Yes	No	No	Yes	-
		MAAC [137]		Off	Yes	No	No	Yes	[138]
		IMPALA [139]		Off	Yes	Yes	Yes	Yes	-
		SEED [140]		Off	Yes	Yes	Yes	Yes	-

Table 3
RL Algorithms

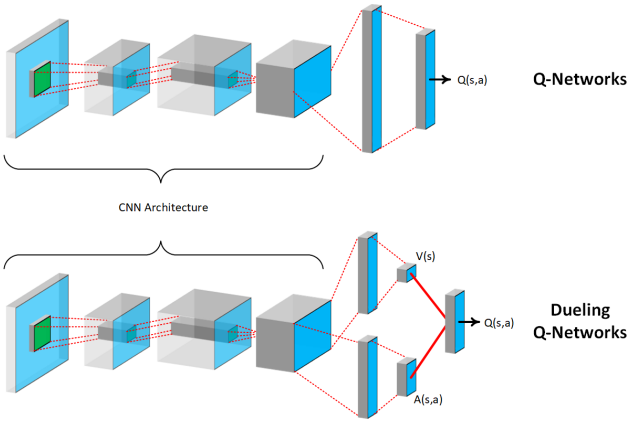

Figure 8: DQN vs. Dueling DQN

it is used to produce the estimated Q-value [81]. The target network parameters are updated after a certain number of time steps by copying the policy network parameters rather than using the backpropagation.

Another improvement on DQN is Dueling DQN illustrated in Figure 8 [89]. Dueling DQN tries to define a better way to evaluate the Q-value by explicitly decomposing the Q-value function into two functions:

- State-Value function $\boldsymbol{V(s)}$ measures how good is for the agent to be in state $\boldsymbol{s}$.
- Advantage-Value function $\boldsymbol{A(s,a)}$ captures how good is an action compared to other actions at a given state.

The two functions shown in Figure 8 are combined via a special aggregation layer to produce an estimate of the state-action value function [89]. The value of this function is equal to the summation of the two values produced by the two functions:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a)\right) \tag{7}$$

The subtracted term $\frac{1}{|\mathcal{A}|}\sum_{a'} \boldsymbol{A(s,a)}$ represents the mean, where $\boldsymbol{|\mathcal{A}|}$ represents the size of the vector $\boldsymbol{A}$. This term helps with identifiability, and it does not change the relative rank of the A (and hence Q) values. Additionally, it increases the stability of the optimization as the advantage function only needs to change as fast as the mean [89].

Double Dueling DQN (DD-DQN) is another DQN algorithm: it combines Dueling DQN with Double DQN to find the optimal Q-value as suggested originally by Wang *et al.* [89], where the output from the Dueling DQN is passed to Double DQN.

Deep Recurrent Q-Network (DRQN) [90] is an extension of the DQN algorithm, replacing the first fully connected layer with a recurrent LSTM layer of the same size. Adding the LSTM layer requires changing the input size from a single state of the environment to multiple states (frames) as a single input, which helps to integrate information through time [90]. The doubling and dueling techniques can be used separately or together in conjunction with recurrent neural network.

6.2.2. Distributional DQN

The goal of distributional Q-learning is to obtain a more accurate representation of the distribution of observed rewards. Fortunato *et al.* [94] introduced NoisyNet, a deep reinforcement learning agent that uses gradient descent to learn parametric noise added to the network weights, and demonstrated how the agent's policy's induced stochasticity can be used to aid efficient exploration [94].

Categorical Deep Q-Networks (C51-DQN) [95] applied a distributional perspective using Wasserstein metric to the random return received by Bellman's equation to approximate value distributions instead of the value function. The algorithm first performs a heuristic projection step and then minimizes the Kullback-Leibler (KL) divergence between the projected Bellman update and the prediction [95].

Quantile Regression Deep Q-Networks (QR-DQN) [96] performs a distributional reinforcement learning over the Wasserstein metric in a stochastic approximation setting. Using Wasserstein distance, the target distribution is minimized by stochastically adjusting the distributions' locations using quantile regression [96]. QR-DQN assigns fixed, uniform probabilities to N adjustable locations and minimizes the quantile Huber loss between the Bellman updated distribution and current return distribution, whereas C51-DQN uses N fixed locations ($N = 51$) for distribution approximation and adjusts the locations probabilities [96].

Implicit Quantile Networks (IQN) [97] incorporates QR-DQN [96] to learn full quantile function controlled by the size of the network and the amount of training, in contrast to QR-DQN quantile function that learns a discrete set of quantiles dependent on the number of quantiles output [97]. IQN distribution function assumes the base distribution to be non-uniform and reparameterizes samples from a base distribution to the respective quantile values of a target distribution.

Rainbow DQN [98] combines several improvements of the traditional DQN algorithm into a single algorithm. These improvements are [98]:

- Addressing the overestimation bias of Q-learning using Double DQN [81].
- Improving data efficiency by replaying more often significant transitions using Prioritized Experience Replay (PER) [6].
- Generalizing across actions by decomposing the Q-value function using Dueling DQN into [89]: 1) state values, and 2) action advantages.
- Shifting the bias-variance trade-off and propagating newly observed rewards faster to earlier visited states as implemented in A3C [129].
- Learning a distributional reinforcement learning instead of the expected return similar to C51-DQN [95].

- Implementing stochastic network layers using Noisy DQN [94].

Yang *et al.* [99] proposed Fully parameterized Quantile Function (FQF) for distributional RL providing full parameterization for both quantile fractions and corresponding quantile values, unlike QR-DQN [96] and IQN [97] where quantile fractions are fixed or sampled, and only the corresponding quantile values are parameterized [99].

FQF for distributional RL uses two networks: 1) quantile value network that maps quantile fractions to corresponding quantile values, and 2) fraction proposal network that generates quantile fractions for each state-action pair with the goal of distribution approximation while minimizing the 1-Wasserstein distance between the approximated and actual distribution [99].

6.2.3. Distributed DQN

Distributed DRL architecture used by different RL algorithms as depicted in Figure 9 [103] aims to decouple acting from learning in distributed settings relaying on prioritized experience replay to focus on the significant experiences generated by actors. The actors share the same NN and replay experience buffer, where they interact with the environment and store their experiences in the shared replay experience buffer. On the other hand, the learner replays prioritized experiences from the shared experience buffer and updates the learner NN accordingly [101]. In theory, both acting and learning can be distributed across multiple workers or running on the same machine [101].

Ape-X DQN [101], based on the Ape-X framework, was the first algorithm to suggest distributed DRL, which was later extended by Recurrent Replay Distributed DQN (R2D2) [100] with two main differences: 1) R2D2 adds an LSTM layer after the convolutional stack to overcome partial observability, and 2) it trains a recurrent neural network from randomly sampled replay sequences using the "burn-in" strategy, which produces a start state through using a portion of the replay sequence and updates the network only on the remaining part of the sequence [100].

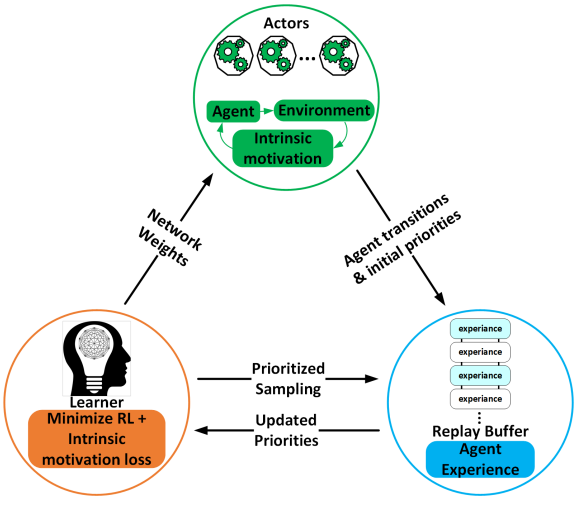

Figure 9: Distributed DRL agent scheme

Never Give Up (NGA) [102] is another algorithm that combines R2D2 architecture with a novel approach that encourages the agent to learn exploratory strategies throughout the training process using a compound intrinsic reward consisting of two modules:

- Life-long novelty module uses Random Network Distillation (RND) [142], which consists of two networks used to generate an intrinsic reward: 1) target network, and 2) prediction network. This mechanism is known as curiosity because it motivates the agent to explore the environment by going to novel or unfamiliar states.
- Episodic novelty module uses dynamically-sized episodic memory M that stores the controllable states in an online fashion, then turns state-action counts into a bonus reward, where the count is computed using the k-nearest neighbors.

While NGA uses intrinsic reward to promote exploration, it promotes exploitation by generating extrinsic reward using the Universal Value Function Approximator (UVFA). NGA uses conditional architecture with shared weights to learn a family of policies that separate exploration and exploitation [102].

Agent57 [103] is the first RL algorithm that outperforms the human benchmark on all 57 games of Atari 2600. Agent57 implements NGA algorithms with the main difference of applying an adaptive mechanism for exploration-exploitation trade-off and utilizes parameterization of the architecture that allows for more consistent and stable learning [103].

6.2.4. Deep SARSA

Basic SARSA uses Q-learning and is suitable for limited states and discrete actions environments, as described in subsection 6.1. On the other hand, Deep SARSA [104] uses a deep neural network similar to DQN and has the same extensions Double SARSA, Dueling SARSA, Double Dueling SARSA (DD-SARSA), Deep Recurrent SARSA (DR-SARSA), and Double Dueling Deep Recurrent (DD-DR-SARSA). The main difference compared to DQN is that Deep SARSA computes $\boldsymbol{Q(s', a')}$ by performing the next action $\boldsymbol{a'}$, which is required to calculate the current state-action $\boldsymbol{Q(s, a)}$.

6.3. Unlimited States and Continuous Actions

Although discrete actions are sufficient to move a car or UAV in a virtual environment, such actions do not provide a realistic object movement in real-life scenarios. Continuous actions describe the quantity of movement in different directions, and the agent does not choose from a list of predefined actions. For example, a realistic UAV movement specifies the quantity of required change in roll, pitch, yaw, and throttle values to navigate the environment while avoiding obstacles, rather than moving UAV using one step in predefined directions: up, down, left, right, and forward.

Continuous action space requires the agent to learn a parameterized policy $\boldsymbol{\pi_\theta}$ to maximize the expected summation of the discounted rewards because it is impossible to calculate action-value for all continuous actions at different states. The problem is a maximization problem and can be solved using gradient descent algorithms to find the optimal $\boldsymbol{\theta}$. The value of $\boldsymbol{\theta}$ is updated as follows:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{8}$$

where $\boldsymbol{\alpha}$ is the learning rate and $\boldsymbol{\nabla}$ is the gradient.

The reward function $\boldsymbol{J}$ objective is to maximize the expected reward using a parameterized policy π_θ as follows [143]:

$$\begin{aligned} J(\pi_\theta) &= \sum_{s \in S} \rho_{\pi_\theta}(s)\, V^{\pi_\theta}(s) \\ &= \sum_{s \in S} \rho_{\pi_\theta}(s) \sum_{a \in A} Q^{\pi_\theta}(s,a)\, \pi_\theta(a|s) \end{aligned} \tag{9}$$

Here $\boldsymbol{\rho_{\pi_\theta}(s)}$ defines the stationary probability of $\boldsymbol{\pi_\theta}$ starting from state $\boldsymbol{s_0}$ and transitioning to future states following the policy $\boldsymbol{\pi_\theta}$ for $\boldsymbol{t}$ time steps. Finding the optimal $\boldsymbol{\theta}$ that maximizes the function $\boldsymbol{J(\pi_\theta)}$ requires finding the gradient $\boldsymbol{\nabla_\theta J(\theta)}$:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \left(\sum_{s \in S} \rho_{\pi_\theta}(s) \sum_{a \in A} Q^{\pi_\theta}(s,a)\, \pi_\theta(a|s) \right) \\ &\propto \sum_{s \in S} \mu(s) \sum_{a \in A} Q^{\pi_\theta}(s,a)\, \nabla \pi_\theta(a|s) \end{aligned} \tag{10}$$

Equation 10 can be further rewritten in continuous episodes since $\boldsymbol{\sum_{s \in S} \eta(s) = 1}$ as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^{\pi_\theta}, a \sim \pi_\theta} \Big[Q^{\pi_\theta}(s,a)\, \nabla_\theta \ln \pi_\theta(a_t|s_t) \Big] \tag{11}$$

Equation 12 [7] known as off-policy gradient theorem shows how the policy is adjusted to the ratio between the target policy $\boldsymbol{\pi_\theta(a|s)}$ and behavior policy $\boldsymbol{\beta(a|s)}$. Notice that the training sample is collected according to the target policy $\boldsymbol{s \sim \rho^{\pi_\theta}}$ and the expected return is generated for the same policy $\boldsymbol{\pi_\theta}$, where the training sample follows a behavior policy $\boldsymbol{\beta(a|s)}$.

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^{\beta}, a \sim \beta} \Big[\frac{\pi_\theta(a|s)}{\beta_\theta(a|s)} Q^{\pi_\theta}(s,a)\, \nabla_\theta \ln \pi_\theta(a_t|s_t) \Big] \tag{12}$$

The policy gradient theorem shown in equation 9 [144] considered the fundamental base of distinct Policy Gradients (PG) algorithms such as REINFORCE [105], Actor-Critic algorithms [145], and different multi-agent and distributed actor-critic algorithms.

6.3.1. Policy-Based Algorithms

Policy-based algorithms focus on improving the performance of gradient descent using different techniques such as REINFORCE [105], Trust Region Policy Optimization (TRPO) [106], Proximal Policy Optimization (PPO) [108], Phasic Policy Gradient (PPG) [115], and Stein Variational Policy Gradient (SVPG) [116].

REINFORCE

REINFORCE is an acronym for **RE**ward **I**ncrement = **N**onnegative **F**actor × **O**ffset **R**einforcement × **C**haracteristic **E**ligibility [105], which is a Monte-Carlo policy gradient algorithm that works with the episodic case. It requires a complete episode to obtain a sample proportional to the gradient and updates the policy parameter $\boldsymbol{\theta}$ with the step size $\boldsymbol{\alpha}$. Because $\boldsymbol{\mathbb{E}_\pi[G_t|S_t, A_t] = Q^\pi(s,a)}$, REINFORCE can be defined as [143]:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \Big[G_t\, \nabla_\theta \ln \pi_\theta(A_t|S_t) \Big] \tag{13}$$

REINFORCE uses the Monte Carlo method, which suffers from high variance and, consequently, has slow learning [105]. Adding a baseline to REINFORCE reduces the variance and speeds up learning while keeping the bias unchanged by subtracting the baseline value from the expected return $\boldsymbol{G_t}$ [143].

Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) [106] is a PG algorithm that improves the performance of gradient descent by taking more extensive steps within trust regions defined by a constraint of KL-Divergence and performs the policy update after each trajectory rather than after each state. Proximal Policy Optimization (PPO) [108] can be considered an extension of TRPO; it imposes the constraint as a penalty and clips the objective to ensure that the optimization is carried out within the predefined range [146].

Phasic Policy Gradient (PPG) [115] extends PPO by including a periodic auxiliary phase which distills features from the value function into the policy network to improve training. This auxiliary phase enables feature sharing between the policy and value function while decoupling their training.

Stein Variational Policy Gradient (SVPG) Stein Variational Policy Gradient (SVPG) [116] applies the Stein variational gradient descent (SVGD) [147] to update the policy parameterized by $\boldsymbol{\theta}$, which reduces variance and improves convergence. SVPG improves the average return and data efficiency when used on top of *REINFORCE* and the *advantage actor-critic* algorithms [116].

6.3.2. Actor-Critic

Actor-critic algorithms are a set of algorithms based on policy gradients theorem that consist of two components:

1. An Actor responsible for adjusting the parameter $\boldsymbol{\theta}$ of the policy $\boldsymbol{\pi_\theta}$

2. A Critic which employs a parameterized vector $\boldsymbol{w}$ to estimate the value-function $\boldsymbol{Q^w(s_t, a_t) \approx Q^\pi(s_t, a_t)}$ using a policy evaluation algorithm such as temporal-difference learning [7]

The actor can be described as the network trying to find the probability of all available actions and select the action with the highest value, while the critic can be described as a network evaluating the selected action by estimating the value of the new state resulted from performing the action. Different algorithms fall under the actor-critic category.

Deterministic Policy Gradients (DPG)

Deterministic policy gradients (DPG) algorithms model the policy as a deterministic policy $\boldsymbol{\mu(s)}$ rather than a stochastic policy $\boldsymbol{\pi(s, a)}$ modeled over the action's probability distribution. We described earlier in Equation 9, the objective function under a selected policy $\boldsymbol{J(\pi_\theta)}$ to be $\boldsymbol{\sum_{s\in S} \rho_{\pi_\theta}(s) \;\; V^{\pi_\theta}(s)}$; however, a deterministic policy is a special case of stochastic policy, where the objective function of the target policy is averaged over the state distribution of the behavior policy as described in equation 14 [7].

$$\begin{aligned} J_\beta(\mu_\theta) &= \int_S \rho^\beta(s)\, V^\mu(s)\, ds \\ &= \int_S \rho^\beta(s)\, Q^\mu(s, \mu_\theta(s))\, ds \end{aligned} \tag{14}$$

In the off-policy approach with a stochastic policy, importance sampling is often used to correct the mismatch between behavior and target policies. However, because the deterministic policy gradient removes the integral over actions, we can avoid importance sampling and the gradient becomes:

$$\begin{aligned} \nabla_\theta J_\beta(\mu_\theta) &\approx \int_S \rho^\beta(s)\, \nabla_\theta\, \mu_\theta(a|s)\, Q^\mu(s, \mu_\theta(s))\, ds \\ &= \mathbb{E}_{s\sim\rho^\beta}\left[\nabla_\theta\, \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}\right] \end{aligned} \tag{15}$$

DPG learns a deterministic policy rather than a stochastic policy, which requires exploration to learn the gradient of $\boldsymbol{Q}$ regarding the actions using backpropagation through time (BPTT). Experience Replay (ER) can be used in conjunction with DPG to improve stability and data efficiency [121]. Different algorithms build on DPG with improvements; for example, Deep Deterministic Policy Gradient (DDPG) [122] adapts DQN to work with continuous action space and combines it with DPG.

Twin Delayed Deep Deterministic (TD3) [126] builds on Double DQN and applies to DDPG to prevent the overestimation of the value function by taking the minimum value between a pair of critics [126].

Recurrent Deterministic Policy Gradients (RDPG)

Wierstra *et al.* [148] applied RNN to Policy Gradient (PG) to build a model-free RL - namely Recurrent Policy Gradient (RPG), for Partially Observable Markov Decision Problem (POMDP), which does not require the agent to have a complete assumption about the environment [148]. RPG applies a method for backpropagating return-weighted characteristic eligibilities through time to approximate a policy gradient for a recurrent neural network [148].

Recurrent Deterministic Policy Gradient (RDPG) [121] implements DPG using RNN and extends the work of RGP [148] to partially observed domains. The RNN with LSTM cells preserves information about past observations over many time steps.

Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) aims to maximize the expected reward while maximizing the entropy [127]. SAC ameliorates the maximum expected sum of rewards defined through accumulating the reward over states transitions $J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s\sim\rho^\pi, a\sim\pi}\Big[r(s_t, a_t)\Big]$ by adding the expected entropy of the policy over $\rho_\pi(s_t)$ [127]. Equation 16 shows a generalized entropy objective, where the temperature parameter α controls the stochasticity of the optimal policy through defining the relevance of the entropy $\mathcal{H}(\pi(.|s_t))$ term to the reward [127].

$$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s\sim\rho^\pi, a\sim\pi}\Big[r(s_t, a_t) + \alpha\mathcal{H}(\pi(.|s_t))\Big] \tag{16}$$

SAC uses two separate neural networks for the actor and critic, and applies function approximators to estimate a soft Q-function $\boldsymbol{Q_\theta(s_t, a_t)}$ parameterized by $\boldsymbol{\theta}$, a state value function $\boldsymbol{V_\psi(s_t)}$ parameterized by $\boldsymbol{\psi}$, and an adjustable policy $\boldsymbol{\pi_\phi(a_t|s_t)}$ parameterized by $\boldsymbol{\phi}$.

6.3.3. Multi-Agent and Distributed Actor-Critic

This group of algorithms includes multi-agent and distributed actor-critic algorithms. They are grouped together as multi-agents can be deployed across several nodes making it a distributed system.

Advantage Actor-Critic

Asynchronous Advantage Actor-Critic (A3C) [129] is a policy gradient algorithm that uses multi-threads, also known as agents or workers, for parallel training. Each agent maintains a local policy $\boldsymbol{\pi_\theta(a_t|s_t)}$ and an estimate of the value function $\boldsymbol{V_\theta(s_t)}$. The agent synchronizes its parameters with the global network having the same structure. The agents work asynchronously, where the value of the network parameters flows in both directions between the agents and the global network. The policy and the value function are updated after t_{max} actions or when a final state is reached [129].

Advantage Actor-Critic (A2C) [129] is a policy gradient algorithm similar to A3C, except it has a coordinator responsible for synchronizing all agents. The coordinator waits for all agents to finish their work either by reaching a final state or by performing $\boldsymbol{t_{max}}$ actions before it updates the policy and the value function in both direction between the agents and the global network.

Actor-Critic using Kronecker-Factored Trust Region (ACKTR) [134] is another extension of A3C [129], which optimizes both the actor and critic by using Kronecker-factored approximation curvature (K-FAC) [149]. It provides an improved computation of the natural gradients by allowing the covariance matrix of the gradient to be efficiently inverted [134].

Actor-Critic with Experience Replay (ACER)

Actor-Critic with Experience Replay (ACER) [132] is an off-policy actor-critic algorithm with experience replay that uses a single deep neural network to estimate the policy $\pi_\theta(a_t|s_t)$ and the value function $V^{\pi}_{\theta_v}(s_t)$ [132]. The three main advantages of ACER over A3C are [129]: 1) it improves the truncated importance sampling with the bias correction, 2) it uses stochastic dueling network architectures, and 3) it applies a new *trust region policy optimization* method [132].

ACER uses an improved Retrace algorithm [150] by applying truncated importance sampling with bias correction technique and using the value $\boldsymbol{Q^{ret}}$ as the target value to train the critic by minimizing the L2 error term [132]. In ACER, the gradient $\hat{\boldsymbol{g}}_{\boldsymbol{t}}^{\boldsymbol{acer}}$ is computed by truncating the importance weights by a constant $\boldsymbol{c}$, and subtracting $\boldsymbol{V_{\theta_v}(s_t)}$ to reduce variance [132].

Retrace-Actor (Reactor) [133] combines contributions from multiple algorithms to improve sampling and timing efficiency. It uses Distributional Retrace [150] to perform distributional RL multi-step off-policy updates while applying prioritized replay to the transitions [133]. Furthermore, Reactor improves the trade-off between variance and bias using β-leave-one-out (β-LOO) leading to policy gradient improvement by using the action values as a baseline [133].

Multi-Agent Reinforcement Learning (MARL)

While Distributed Distributional DDPG (D4PG) [128] adopts distributed settings for DDPG with additional improvements such as using N-step returns and prioritized experience replay [128], Multi-Agent DDPG (MADDPG) [135] extends DDPG to work with multi-agents, where it considers action policies of other agents and learns policies that require multi-agent coordination [135]. In contrast, Multi-Agent TD3 (MATD3) [136] extends TD3 to multi-agent settings using centralized training and decentralized execution. Similarly to TD3, MATD3 uses two centralized critics for each agent to address the overestimation bias produced by MADDPG.

Importance Weighted Actor-Learner Architecture (IMPALA) [139] is an off-policy learning algorithm that decouples acting from learning and can be used in two different setups: 1) single learner and 2) multiple synchronous learners.

Using a single learner and multiple actor setup, each actor generates trajectories, sends each trajectory to the learner, and receives the updated policy before starting a new trajectory. The learner learns from the actors simultaneously by saving the received trajectories from the actors in a queue and generating the updated policy. Nevertheless, actors might learn an older model because they are not aware of each other and because of the lag between the actors and the learner. To resolve this issue, IMPALA uses a novel v-trace correction method that considers a truncated importance sampling (IS), which is the ratio between the learner policy $\boldsymbol{\pi}$ and the actor current policy $\boldsymbol{\mu}$. Similarly, in multiple synchronous learners, the policy parameters are distributed across multiple learners that work synchronously through a master learner [139].

Scalable, Efficient Deep-RL (SEED RL) [140] provides a scalable architecture that combines IMPALA with R2D2 and can train on millions of frames per second with a lower cost of experiments compared to IMPALA. SEED moves the inference to the learner while the environments run remotely, introducing a latency issue due to the increased number of remote calls, which is mitigated using a fast communication layer using gRPC.

7. Challenges and Opportunities

Previous sections demonstrated the diversity of UAV navigation tasks besides the diversity of RL algorithms. Due to such a high number of algorithms, selecting the appropriate algorithms for the task at hand is challenging. Table 3 and discussion in Section 6 provide an overview of the RL algorithms and assist in selecting the RL algorithm for navigation task. Nevertheless, there are still numerous challenges and opportunities in RL for UAV navigation, including:

Evaluation and benchmarking: Atari 2600 is a home video game console with 57 built-in games that laid the foundation to establish a benchmarking for RL algorithms. The benchmark was established using 57 different games to compare various RL algorithms and set a benchmark baseline against human performance playing the same games. The agent's performance is evaluated and compared to other algorithms using the same benchmark (Atari 2600). The performance of the algorithms using the benchmark might differ when applied to the UAV navigation simulated on a 3D environment or the real world because the games in Atari 2600 can provide a full state of the environment, which means the agent does not need to make assumptions about the state, and MDP can be applied to these problems. Whereas in UAV navigation simulation, the agent knows partial states of the environment (observation), these observations are used to train the agent using POMDP, which results in changing behavior for some of the algorithms. Furthermore, images processed from the games are 2D images, where the agent in most algorithms tries to learn an optimal policy based on the pattern of the pixels in the image. The same cannot be inferred for images received from the 3D simulators or the real-world images because objects' depth plays a vital role in learning the optimal policy avoiding nearby objects. Therefore, there is a need for new evaluation and benchmarking techniques for RL driven navigation.

Environment complexity: The tendency to oversimplify the environment and the absence of a standardized benchmarking tools makes it impossible to compare and conclude

performances obtained using different algorithms and simulated using various tools and environments. Nevertheless, the UAV needs to perform tasks in different environments and is subject to various conditions, for example:

- Navigating in various environment types such as indoor vs. outdoor.
- Considering the changing environment conditions such as wind speed, lighting conditions, and moving objects.

Some of the simulation tools discussed in Section 5, such as AirSim combined with Unreal Engine, provide different environment types out-of-the-box and are capable of simulating several environmental effects such as changing wind speed and lighting conditions. Still, these complex environments remain to be combined with new benchmarking techniques for improved comparison of RL algorithms for UAV navigation.

Knowledge Transfer Knowledge transfer imposes another challenge, where the RL agent training in a selected environment does not guarantee similar performance in another environment due to the difference in environments' nature such as different object/obstacles types, background texture, lighting density, and added noise. Most of the existing research focused on applying transfer learning to reduce the training time for the agent in the new environment [51]. However, generalized training methods or other techniques are needed to guarantee a similar performance of the agent in different environments and under various conditions.

Algorithm diversity: As seen from Table 3, many recent and very successful algorithms have not been applied in UAV navigation. As these algorithms have shown great surcease in other domains outperforming the human benchmark, there is a prodigious potential in their application in UAV navigation. The algorithms are expected to gain better generalization on different environments, speed up the training process, and even solve efficiently more complex tasks such as UAVs flocking.

8. Conclusion

This review deliberates on the application of RL for autonomous UAV navigation. RL uses an intelligent agent to control the UAV movement by processing the states from the environment and moving the UAV in desired directions. The data received from the UAV camera or other sensors such as LiDAR are used to estimate the distance from various objects in the environment and avoid colliding with these objects.

RL algorithms and techniques were used to solve navigation problems such as controlling the UAV while avoiding obstacles, path planning, and flocking. For example, RL is used in single UAV path planning and multi-UAVs flocking to plan path waypoints of the UAV(s) while avoiding obstacles or maintaining flight formation (flocking). Furthermore, this study recognizes various navigation frameworks simulation software used to conduct the experiments along with identifying their use within the reviewed papers.

The review discusses over fifty RL algorithms, explains their contributions and relations, classifies them according to the application environment and their use in UAV navigation. Furthermore, the study highlights other algorithmic traits such as multi-threading, distributed processing, and multi-agents.

The study observes that the research community tends to experiment with a specific set of algorithms: Q-learning, DQN, Double DQN, DDPG, PPO, although some recent algorithms show more promising results than the mentioned algorithms such as agent57. To the best of the authors' knowledge, this study is the first systematic review identifying a large number of RL algorithms while focusing on their application in autonomous UAV navigation.

Analysis of the current RL algorithms and their use in UAV navigation identified the following challenges and opportunities: the need for navigation-focused evaluation and benchmarking techniques, the necessity to work with more complex environments, and the need to examine knowledge transfer and evaluate state-of-the-art RL algorithms as Agent57.

A. Acronyms

- **A2C :** Advantage Actor-Critic
- **A3C :** Asynchronous Advantage Actor-Critic
- **AC :** Actor-Critic
- **ACE :** Actor Ensemble
- **ACER :** Actor-Critic with Experience Replay
- **ACKTR :** Actor-Critic using Kronecker-Factored Trust Region
- **Agent57 :** Agent57
- **Ape-X DPG :** Ape-X Deterministic Policy Gradients
- **Ape-X DQN :** Ape-X Deep Q-Networks
- **C51-DQN :** Categorical Deep Q-Networks
- **D4PG :** Distributed Distributional DDPG
- **DAC :** Double Actor-Critic
- **DD-DQN :** Double Dueling Deep Q-Networks
- **DD-DRQN :** Double Dueling Deep Recurrent Q-Networks
- **DDPG :** Deep Deterministic Policy Gradient
- **Double DQN :** Double Deep Q-Networks
- **DPG :** Deterministic Policy Gradients
- **DPPO :** Distributed Proximal Policy Optimization
- **DQN :** Deep Q-Networks
- **DRL :** Deep Reinforcement Learning
- **DRQN :** Deep Recurrent Q-Networks
- **Dueling DQN :** Dueling Deep Q-Networks
- **FQF :** Fully parameterized Quantile Function
- **GPS :** Global Positioning System
- **IMPALA :** Importance Weighted Actor-Learner Architecture
- **IMU :** Inertial Measurement Unit
- **IQN :** Implicit Quantile Networks

- **K-FAC :** Kronecker-factored approximation curvature
- **MAAC :** Multi-Actor-Attention-Critic
- **MADDPG :** Multi-Agent DDPG
- **MARL :** Multi-Agent Reinforcement Learning
- **MATD3 :** Multi-Agent Twin Delayed Deep Deterministic
- **MATD3 :** Multi-Agent TD3
- **MDP :** Markov Decision Problem
- **NGA :** Never Give Up
- **Noisy DQN :** Noisy Deep Q-Networks
- **PAAC :** Parallel Advantage Actor-Critic
- **PG :** Policy Gradients
- **POMDP :** Partially Observable Markov Decision Problem
- **PPG :** Phasic Policy Gradient
- **PPO :** Proximal Policy Optimization
- **QR-DQN :** Quantile Regression Deep Q-Networks
- **R2D2 :** Recurrent Replay Distributed Deep Q-Networks
- **Rainbow DQN :** Rainbow Deep Q-Networks
- **RDPG :** Recurrent Deterministic Policy Gradients
- **Reactor :** Retrace-Actor
- **REINFORCE : RE**ward **I**ncrement = **N**onnegative **F**actor × **O**ffset **R**einforcement × **C**haracteristic **E**ligibility
- **RL :** Reinforcement Learning
- **RND :** Random Network Distillation
- **SAC :** Soft Actor-Critic
- **SARSA :** State-Action-Reward-State-Action
- **SEED RL :** Scalable, Efficient Deep-RL
- **SLAC :** Stochastic Latent Actor-Critic
- **SVPG :** Stein Variational Policy Gradient
- **TD3 :** Twin Delayed Deep Deterministic
- **TRPO :** Trust Region Policy Optimization
- **UVFA :** Universal Value Function Approximator

CRediT authorship contribution statement

Fadi AlMahamid: Conceptualization, Methodology, Investigation, Formal Analysis, Validation, Writing - Original Draft, Writing - Review & Editing. **Katarina Grolinger:** Supervision, Writing - Review & Editing, Funding Acquisition.

Acknowledgments

This research has been supported by NSERC under grant RGPIN-2018-06222

References

[1] Y. Lu, Z. Xue, G.-S. Xia, L. Zhang, A survey on vision-based uav navigation, Taylor & Francis Geo-spatial information science 21 (1) (2018) 21–32. doi:10.1080/10095020.2017.1420509.

[2] F. Zeng, C. Wang, S. S. Ge, A survey on visual navigation for artificial agents with deep reinforcement learning, IEEE Access 8 (2020) 135426–135442.

[3] R. Azoulay, Y. Haddad, S. Reches, Machine learning methods for uav flocks management-a survey, IEEE Access 9 (2021) 139146–139175.

[4] S. Aggarwal, N. Kumar, Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges, Computer Communications 149 (2020) 270–299.

[5] L.-J. Lin, Self-improving reactive agents based on reinforcement learning, planning and teaching, Springer Machine learning 8 (3-4) (1992) 293–321.

[6] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, arXiv:1511.05952 (2015).

[7] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: PMLR International conference on machine learning, 2014, pp. 387–395.

[8] A. Chapman, Drone Types: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL, https://www.auav.com.au/articles/drone-types/, (Accessed: 01.11.2021) (2016).

[9] N. Salvatore, S. Mian, C. Abidi, A. D. George, A neuro-inspired approach to intelligent collision avoidance and navigation, in: IEEE Digital Avionics Systems Conference, 2020, pp. 1–9.

[10] M. Elnaggar, N. Bezzo, An irl approach for cyber-physical attack intention prediction and recovery, in: IEEE American Control Conference, 2018, pp. 222–227.

[11] W. Andrew, C. Greatwood, T. Burghardt, Deep learning for exploration and recovery of uncharted and dynamic targets from uav-like vision, in: IEEE RSJ International Conference on Intelligent Robots and Systems, 2018, pp. 1124–1131.

[12] Z. Cui, Y. Wang, Uav path planning based on multi-layer reinforcement learning technique, IEEE Access 9 (2021) 59486–59497.

[13] R. B. Grando, J. C. de Jesus, P. L. Drews-Jr, Deep reinforcement learning for mapless navigation of unmanned aerial vehicles, in: IEEE Latin American Robotics Symposium, Brazilian Symposium on Robotics and Workshop on Robotics in Education, 2020, pp. 1–6.

[14] C. Wang, J. Wang, J. Wang, X. Zhang, Deep-reinforcement-learning-based autonomous uav navigation with sparse rewards, IEEE Internet of Things Journal 7 (7) (2020) 6180–6190.

[15] C. H. Liu, X. Ma, X. Gao, J. Tang, Distributed energy-efficient multi-uav navigation for long-term communication coverage by deep reinforcement learning, IEEE Transactions on Mobile Computing 19 (6) (2019) 1274–1285.

[16] P. Yan, C. Bai, H. Zheng, J. Guo, Flocking control of uav swarms with deep reinforcement leaming approach, in: IEEE International Conference on Unmanned Systems, 2020, pp. 592–599.

[17] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, E. A. Theodorou, Information theoretic mpc for model-based reinforcement learning, in: IEEE International Conference on Robotics and Automation, 2017, pp. 1714–1721.

[18] S. Omi, H.-S. Shin, A. Tsourdos, J. Espeland, A. Buchi, Introduction to uav swarm utilization for communication on the move terminals tracking evaluation with reinforcement learning technique, in: IEEE European Conference on Antennas and Propagation, 2021, pp. 1–5.

[19] D. Sacharny, T. C. Henderson, Optimal policies in complex large-scale uas traffic management, in: IEEE International Conference on Industrial Cyber Physical Systems, 2019, pp. 352–357.

[20] Z. Yijing, Z. Zheng, Z. Xiaoyi, L. Yang, Q learning algorithm based uav path learning and obstacle avoidence approach, in: IEEE Chinese Control Conference, 2017, pp. 3397–3402.

[21] H. X. Pham, H. M. La, D. Feil-Seifer, L. Van Nguyen, Reinforcement learning for autonomous uav navigation using function approximation, in: IEEE International Symposium on Safety, Security, and Rescue Robotics, 2018, pp. 1–6.

[22] A. Guerra, F. Guidi, D. Dardari, P. M. Djuric, Reinforcement learning for uav autonomous navigation, mapping and target detection, in: ION Position, Location and Navigation Symposium, 2020, pp. 1004–1013.

[23] V. A. Bakale, Y. K. VS, V. C. Roodagi, Y. N. Kulkarni, M. S. Patil, S. Chickerur, Indoor navigation with deep reinforcement learning, in: IEEE International Conference on Inventive Computation Technologies, 2020, pp. 660–665.

[24] B. Zhou, W. Wang, Z. Liu, J. Wang, Vision-based navigation of uav with continuous action space using deep reinforcement learning, in:

IEEE Chinese Control And Decision Conference, 2019, pp. 5030–5035.

[25] H. D. Escobar-Alvarez, N. Johnson, T. Hebble, K. Klingebiel, S. A. Quintero, J. Regenstein, N. A. Browning, R-advance: Rapid adaptive prediction for vision-based autonomous navigation, control, and evasion, WOL Journal of Field Robotics 35 (1) (2018) 91–100. doi:10.1002/rob.21744.

[26] C. W. Reynolds, Flocks, herds and schools: A distributed behavioral model, in: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, 1987, pp. 25–34.

[27] R. Olfati-Saber, Flocking for multi-agent dynamic systems: Algorithms and theory, IEEE Transactions on automatic control 51 (3) (2006) 401–420.

[28] G. T. Lee, C. O. Kim, Autonomous control of combat unmanned aerial vehicles to evade surface-to-air missiles using deep reinforcement learning, IEEE Access 8 (2020) 226724–226736.

[29] H. M. La, W. Sheng, Flocking control of multiple agents in noisy environments, in: IEEE International Conference on Robotics and Automation, 2010, pp. 4964–4969.

[30] Y. Jia, J. Du, W. Zhang, L. Wang, Three-dimensional leaderless flocking control of large-scale small unmanned aerial vehicles, Elsevier IFAC-PapersOnLine 50 (1) (2017) 6208–6213.

[31] H. Su, X. Wang, Z. Lin, Flocking of multi-agents with a virtual leader, IEEE transactions on automatic control 54 (2) (2009) 293–307.

[32] S. A. Quintero, G. E. Collins, J. P. Hespanha, Flocking with fixed-wing uavs for distributed sensing: A stochastic optimal control approach, in: IEEE American Control Conference, 2013, pp. 2025–2031.

[33] S.-M. Hung, S. N. Givigi, A q-learning approach to flocking with uavs in a stochastic environment, IEEE transactions on cybernetics 47 (1) (2016) 186–197.

[34] C. Wang, J. Wang, X. Zhang, A deep reinforcement learning approach to flocking and navigation of uavs in large-scale complex environments, in: IEEE Global Conference on Signal and Information Processing, 2018, pp. 1228–1232.

[35] K. Morihiro, T. Isokawa, H. Nishimura, M. Tomimasu, N. Kamiura, N. Matsui, Reinforcement learning scheme for flocking behavior emergence, Journal of Advanced Computational Intelligence and Intelligent Informatics 11 (2) (2007) 155–161.

[36] Z. Xu, Y. Lyu, Q. Pan, J. Hu, C. Zhao, S. Liu, Multi-vehicle flocking control with deep deterministic policy gradient method, in: IEEE International Conference on Control and Automation, 2018, pp. 306–311.

[37] Y. Zhao, J. Guo, C. Bai, H. Zheng, Reinforcement learning-based collision avoidance guidance algorithm for fixed-wing uavs, Hindawi Complexity 2021 (2021). doi:10.1155/2021/8818013.

[38] O. Bouhamed, X. Wan, H. Ghazzai, Y. Massoud, A ddpg-based approach for energy-aware uav navigation in obstacle-constrained environment, in: IEEE World Forum on Internet of Things, 2020, pp. 1–6.

[39] N. Imanberdiyev, C. Fu, E. Kayacan, I.-M. Chen, Autonomous navigation of uav by using real-time model-based reinforcement learning, in: IEEE International conference on control, automation, robotics and vision, 2016, pp. 1–6.

[40] O. Walker, F. Vanegas, F. Gonzalez, S. Koenig, A deep reinforcement learning framework for uav navigation in indoor environments, in: IEEE Aerospace Conference, 2019, pp. 1–14.

[41] O. Bouhamed, H. Ghazzai, H. Besbes, Y. Massoud, Autonomous uav navigation: A ddpg-based deep reinforcement learning approach, in: IEEE International Symposium on Circuits and Systems, 2020, pp. 1–5.

[42] Y. Li, M. Li, A. Sanyal, Y. Wang, Q. Qiu, Autonomous uav with learned trajectory generation and control, in: IEEE International Workshop on Signal Processing Systems, 2019, pp. 115–120.

[43] J. Zhang, Z. Yu, S. Mao, S. C. Periaswamy, J. Patton, X. Xia, Iadrl: Imitation augmented deep reinforcement learning enabled ugv-uav coalition for tasking in complex environments, IEEE Access 8 (2020) 102335–102347.

[44] E. Camci, E. Kayacan, Planning swift maneuvers of quadcopter using motion primitives explored by reinforcement learning, in: IEEE American Control Conference, 2019, pp. 279–285.

[45] H. Eslamiat, Y. Li, N. Wang, A. K. Sanyal, Q. Qiu, Autonomous waypoint planning, optimal trajectory generation and nonlinear tracking control for multi-rotor uavs, in: IEEE European Control Conference, 2019, pp. 2695–2700.

[46] O. Bouhamed, H. Ghazzai, H. Besbes, Y. Massoud, A generic spatiotemporal scheduling for autonomous uavs: A reinforcement learning-based approach, IEEE Open Journal of Vehicular Technology 1 (2020) 93–106.

[47] A. Majd, A. Ashraf, E. Troubitsyna, M. Daneshtalab, Integrating learning, optimization, and prediction for efficient navigation of swarms of drones, in: IEEE Euromicro International Conference on Parallel, Distributed and Network-based Processing, 2018, pp. 101–108.

[48] L. He, N. Aouf, J. F. Whidborne, B. Song, Integrated moment-based lgmd and deep reinforcement learning for uav obstacle avoidance, in: IEEE International Conference on Robotics and Automation, 2020, pp. 7491–7497.

[49] A. Singla, S. Padakandla, S. Bhatnagar, Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge, IEEE Transactions on Intelligent Transportation Systems (2019).

[50] M. A. Akhloufi, S. Arola, A. Bonnet, Drones chasing drones: Reinforcement learning and deep search area proposal, MDPI Drones 3 (3) (2019). doi:10.3390/drones3030058.

[51] I. Yoon, M. A. Anwar, R. V. Joshi, T. Rakshit, A. Raychowdhury, Hierarchical memory system with stt-mram and sram to support transfer and real-time reinforcement learning in autonomous drones, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 9 (3) (2019) 485–497.

[52] O. Robotics, Ros home page, https://www.ros.org/, (Accessed: 01.11.2021) (2021).

[53] M. Research, Microsoft airsim home page, https://microsoft.github.io/AirSim/, (Accessed: 01.11.2021) (2021).

[54] O. S. R. Foundation, Gazebo home page, https://gazebosim.org/, (Accessed: 01.11.2021) (2021).

[55] E. Games, Epic games unreal engine home page, https://www.unrealengine.com, (Accessed: 01.11.2021) (2021).

[56] F. AlMahamid, K. Grolinger, Reinforcement learning algorithms: An overview and classification, in: IEEE Canadian Conference on Electrical and Computer Engineering, 2021, pp. 1–7.

[57] C. J. Watkins, P. Dayan, Q-learning, Springer Machine learning 8 (3-4) (1992) 279–292. doi:10.1007/978-1-4615-3618-5_4.

[58] X. Yu, Y. Wu, X.-M. Sun, A navigation scheme for a random maze using reinforcement learning with quadrotor vision, in: IEEE European Control Conference, 2019, pp. 518–523.

[59] H. Li, S. Wu, P. Xie, Z. Qin, B. Zhang, A path planning for one uav based on geometric algorithm, in: IEEE CSAA Guidance, Navigation and Control Conference, 2018, pp. 1–5.

[60] P. Karthik, K. Kumar, V. Fernandes, K. Arya, Reinforcement learning for altitude hold and path planning in a quadcopter, in: IEEE International Conference on Control, Automation and Robotics, 2020, pp. 463–467.

[61] O. Bouhamed, H. Ghazzai, H. Besbes, Y. Massoud, A uav-assisted data collection for wireless sensor networks: Autonomous navigation and scheduling, IEEE Access 8 (2020) 110446–110460.

[62] S. Kulkarni, V. Chaphekar, M. M. U. Chowdhury, F. Erden, I. Guvenc, Uav aided search and rescue operation using reinforcement learning, in: IEEE SoutheastCon, Vol. 2, 2020, pp. 1–8.

[63] A. Fotouhi, M. Ding, M. Hassan, Deep q-learning for two-hop communications of drone base stations, MDPI Sensors 21 (6) (2021). doi:10.3390/s21061960.

[64] C. Greatwood, A. G. Richards, Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control, Springer Autonomous Robots 43 (7) (2019) 1681–1693.

[65] G. A. Rummery, M. Niranjan, On-line Q-learning using connectionist systems, Vol. 37, University of Cambridge, 1994.

[66] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv:1312.5602 (2013).

[67] H. Huang, J. Gu, Q. Wang, Y. Zhuang, An autonomous uav navigation system for unknown flight environment, in: IEEE International Conference on Mobile Ad-Hoc and Sensor Networks, 2019, pp. 63–68.

[68] S.-Y. Shin, Y.-W. Kang, Y.-G. Kim, Automatic drone navigation in realistic 3d landscapes using deep reinforcement learning, in: IEEE International Conference on Control, Decision and Information Technologies, 2019, pp. 1072–1077.

[69] Y. Chen, N. González-Prelcic, R. W. Heath, Collision-free uav navigation with a monocular camera using deep reinforcement learning, in: IEEE International Workshop on Machine Learning for Signal Processing, 2020, pp. 1–6.

[70] S. F. Abedin, M. S. Munir, N. H. Tran, Z. Han, C. S. Hong, Data freshness and energy-efficient uav navigation optimization: A deep reinforcement learning approach, IEEE Transactions on Intelligent Transportation Systems (2020).

[71] E. Camci, D. Campolo, E. Kayacan, Deep reinforcement learning for motion planning of quadrotors using raw depth images, Learning (RL) 10 (2020).

[72] H. Huang, Y. Yang, H. Wang, Z. Ding, H. Sari, F. Adachi, Deep reinforcement learning for uav navigation through massive mimo technique, IEEE Transactions on Vehicular Technology 69 (1) (2019) 1117–1121.

[73] E. Cetin, C. Barrado, G. Muñoz, M. Macias, E. Pastor, Drone navigation and avoidance of obstacles through deep reinforcement learning, in: IEEE Digital Avionics Systems Conference, 2019, pp. 1–7.

[74] T.-C. Wu, S.-Y. Tseng, C.-F. Lai, C.-Y. Ho, Y.-H. Lai, Navigating assistance system for quadcopter with deep reinforcement learning, in: IEEE International Cognitive Cities Conference, 2018, pp. 16–19.

[75] B. Zhou, W. Wang, Z. Wang, B. Ding, Neural q learning algorithm based uav obstacle avoidance, in: IEEE CSAA Guidance, Navigation and Control Conference, 2018, pp. 1–6.

[76] A. Walvekar, Y. Goel, A. Jain, S. Chakrabarty, A. Kumar, Vision based autonomous navigation of quadcopter using reinforcement learning, in: IEEE International Conference on Automation, Electronics and Electrical Engineering, 2019, pp. 160–165.

[77] A. Viseras, M. Meissner, J. Marchal, Wildfire front monitoring with multiple uavs using deep q-learning, IEEE Access (2021).

[78] E. Camci, E. Kayacan, Learning motion primitives for planning swift maneuvers of quadrotor, Springer Autonomous Robots 43 (7) (2019) 1733–1745.

[79] Á. Madridano, A. Al-Kaff, P. Flores, D. Martín, A. de la Escalera, Software architecture for autonomous and coordinated navigation of uav swarms in forest and urban firefighting, MDPI Applied Sciences 11 (3) (2021). doi:10.3390/app11031258.

[80] A. Bonnet, M. A. Akhloufi, Uav pursuit using reinforcement learning, in: SPIE Unmanned Systems Technology XXI, Vol. 11021, International Society for Optics and Photonics, 2019, pp. 51 – 58. doi:10.1117/12.2520310.

[81] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-Learning, AAAI Conference on Artificial Intelligence (2016) 2094–2100.

[82] S. Zhou, B. Li, C. Ding, L. Lu, C. Ding, An efficient deep reinforcement learning framework for uavs, in: IEEE International Symposium on Quality Electronic Design, 2020, pp. 323–328.

[83] A. Anwar, A. Raychowdhury, Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning, IEEE Access 8 (2020) 26549–26560.

[84] Y. Yang, K. Zhang, D. Liu, H. Song, Autonomous uav navigation in dynamic environments with double deep q-networks, in: IEEE Digital Avionics Systems Conference, 2020, pp. 1–7.

[85] M. A. Anwar, A. Raychowdhury, Navren-rl: Learning to fly in real environment via end-to-end deep reinforcement learning using monocular images, in: IEEE International Conference on Mechatronics and Machine Vision in Practice, 2018, pp. 1–6.

[86] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, A. Cangelosi, Toward end-to-end control for uav autonomous landing via deep reinforcement learning, in: IEEE International conference on unmanned aircraft systems, 2018, pp. 115–123.

[87] R. Polvara, S. Sharma, J. Wan, A. Manning, R. Sutton, Autonomous vehicular landings on the deck of an unmanned surface vehicle using deep reinforcement learning, Cambridge Core Robotica 37 (11) (2019) 1867–1882. doi:10.1017/S0263574719000316.

[88] G. Muñoz, C. Barrado, E. Çetin, E. Salami, Deep reinforcement learning for drone delivery, MDPI Drones 3 (3) (2019). doi:10.3390/drones3030072.

[89] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas, Dueling Network Architectures for Deep Reinforcement Learning, in: PMLR International Conference on Machine Learning, Vol. 48, 2016, pp. 1995–2003.

[90] M. Hausknecht, P. Stone, Deep recurrent q-learning for partially observable mdps, arXiv:1507.06527 (2015).

[91] A. Peake, J. McCalmon, Y. Zhang, B. Raiford, S. Alqahtani, Wilderness search and rescue missions using deep reinforcement learning, in: IEEE International Symposium on Safety, Security, and Rescue Robotics, 2020, pp. 102–107.

[92] G. Tong, N. Jiang, L. Biyue, Z. Xi, W. Ya, D. Wenbo, Uav navigation in high dynamic environments: A deep reinforcement learning approach, Elsevier Chinese Journal of Aeronautics 34 (2) (2021) 479–489.

[93] A. Villanueva, A. Fajardo, Deep reinforcement learning with noise injection for uav path planning, in: IEEE International Conference on Engineering Technologies and Applied Sciences, 2019, pp. 1–6.

[94] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al., Noisy networks for exploration, arXiv:1706.10295 (2017).

[95] M. G. Bellemare, W. Dabney, R. Munos, A distributional perspective on reinforcement learning, in: PMLR International Conference on Machine Learning, 2017, pp. 449–458.

[96] W. Dabney, M. Rowland, M. G. Bellemare, R. Munos, Distributional reinforcement learning with quantile regression, AAAI Conference on Artificial Intelligence (2018).

[97] W. Dabney, G. Ostrovski, D. Silver, R. Munos, Implicit quantile networks for distributional reinforcement learning, in: PMLR International conference on machine learning, 2018, pp. 1096–1105.

[98] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver, Rainbow: Combining improvements in deep reinforcement learning, AAAI Conference on Artificial Intelligence (2018).

[99] D. Yang, L. Zhao, Z. Lin, T. Qin, J. Bian, T.-Y. Liu, Fully parameterized quantile function for distributional reinforcement learning, Advances in Neural Information Processing Systems 32 (2019) 6193–6202.

[100] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, W. Dabney, Recurrent experience replay in distributed reinforcement learning, International Conference on Learning Representations (2018).

[101] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, D. Silver, Distributed prioritized experience replay, arXiv:1803.00933 (2018).

[102] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt, et al., Never give up: Learning directed exploration strategies, arXiv:2002.06038 (2020).

[103] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, C. Blundell, Agent57: Outperforming the atari human benchmark, in: PMLR International Conference on Machine Learning, 2020, pp. 507–517.

[104] D. Zhao, H. Wang, K. Shao, Y. Zhu, Deep reinforcement learning with experience replay based on sarsa, in: IEEE Symposium Series

on Computational Intelligence, 2016, pp. 1–6.
[105] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Springer Machine learning 8 (3-4) (1992) 229–256.
[106] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: PMLR International Conference on Machine Learning, Vol. 37, 2015, pp. 1889–1897.
[107] W. Koch, R. Mancuso, R. West, A. Bestavros, Reinforcement learning for uav attitude control, ACM Transactions on Cyber-Physical Systems 3 (2) (2019) 1–21.
[108] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv:1707.06347 (2017).
[109] S. D. Morad, R. Mecca, R. P. Poudel, S. Liwicki, R. Cipolla, Embodied visual navigation with automatic curriculum learning in real environments, IEEE Robotics and Automation Letters 6 (2) (2021) 683–690.
[110] M. Hasanzade, E. Koyuncu, A dynamically feasible fast replanning strategy with deep reinforcement learning, Springer Journal of Intelligent & Robotic Systems 101 (1) (2021) 1–17.
[111] Z. Wang, H. Li, Z. Wu, H. Wu, A pretrained proximal policy optimization algorithm with reward shaping for aircraft guidance to a moving destination in three-dimensional continuous space, SAGE International Journal of Advanced Robotic Systems 18 (1) (2021). doi:10.1177/1729881421989546.
[112] V. J. Hodge, R. Hawkins, R. Alexander, Deep reinforcement learning for drone navigation using sensor data, Springer Neural Computing and Applications 33 (6) (2021) 2015–2033.
[113] A. M. Deshpande, R. Kumar, A. A. Minai, M. Kumar, Developmental reinforcement learning of control policy of a quadcopter uav with thrust vectoring rotors, in: ASME Dynamic Systems and Control Conference, Vol. 84287, 2020, p. V002T36A011. doi:10.1115/DSCC2020-3319.
[114] C. J. Maxey, E. J. Shamwell, Navigation and collision avoidance with human augmented supervisory training and fine tuning via reinforcement learning, in: SPIE Micro-and Nanotechnology Sensors, Systems, and Applications XI, Vol. 10982, 2019, pp. 325 – 334. doi:10.1117/12.2518551.
[115] K. Cobbe, J. Hilton, O. Klimov, J. Schulman, Phasic policy gradient, arXiv:2009.04416 (2020).
[116] Y. Liu, P. Ramachandran, Q. Liu, J. Peng, Stein variational policy gradient, arXiv:1704.02399 (2017).
[117] A. X. Lee, A. Nagabandi, P. Abbeel, S. Levine, Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model, arXiv:1907.00953 (2019).
[118] S. Zhang, H. Yao, Ace: An actor ensemble algorithm for continuous control with tree search, in: AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 5789–5796. doi:10.1609/aaai.v33i01.33015789.
[119] S. Zhang, S. Whiteson, Dac: The double actor-critic architecture for learning options, arXiv:1904.12691 (2019).
[120] S. Li, P. Durdevic, Z. Yang, Optimal tracking control based on integral reinforcement learning for an underactuated drone, Elsevier IFAC-PapersOnLine 52 (8) (2019) 194–199.
[121] N. Heess, J. J. Hunt, T. P. Lillicrap, D. Silver, Memory-based control with recurrent neural networks, arXiv:1512.04455 (2015).
[122] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv:1509.02971 (2015).
[123] D. Wang, T. Fan, T. Han, J. Pan, A two-stage reinforcement learning approach for multi-uav collision avoidance under imperfect sensing, IEEE Robotics and Automation Letters 5 (2) (2020) 3098–3105.
[124] O. Doukhi, D.-J. Lee, Deep reinforcement learning for end-to-end local motion planning of autonomous aerial robots in unknown outdoor environments: Real-time flight experiments, MDPI Sensors 21 (7) (2021) 2534. doi:10.3390/s21072534.
[125] S. Lee, T. Shim, S. Kim, J. Park, K. Hong, H. Bang, Vision-based autonomous landing of a multi-copter unmanned aerial vehicle using reinforcement learning, in: IEEE International Conference on Unmanned Aircraft Systems, 2018, pp. 108–114.
[126] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: PMLR International Conference on Machine Learning, 2018, pp. 1587–1596.
[127] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: PMLR International Conference on Machine Learning, 2018, pp. 1861–1870.
[128] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, T. Lillicrap, Distributed distributional deterministic policy gradients, arXiv:1804.08617 (2018).
[129] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: PMLR International Conference on Machine Learning, 2016, pp. 1928–1937.
[130] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, et al., Emergence of locomotion behaviours in rich environments. arxiv 2017, arXiv:1707.02286 (2017).
[131] C. Alfredo, C. Humberto, C. Arjun, Efficient parallel methods for deep reinforcement learning, in: The Multi-disciplinary Conference on Reinforcement Learning and Decision Making, 2017, pp. 1–6.
[132] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, N. de Freitas, Sample efficient actor-critic with experience replay, arXiv:1611.01224 (2016).
[133] A. Gruslys, W. Dabney, M. G. Azar, B. Piot, M. Bellemare, R. Munos, The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning, arXiv:1704.04651 (2017).
[134] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, J. Ba, Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation, in: Advances in Neural Information Processing Systems, 2017, pp. 5279–5288.
[135] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, arXiv:1706.02275 (2017).
[136] J. Ackermann, V. Gabler, T. Osa, M. Sugiyama, Reducing overestimation bias in multi-agent domains using double centralized critics, arXiv:1910.01465 (2019).
[137] S. Iqbal, F. Sha, Actor-attention-critic for multi-agent reinforcement learning, in: PMLR International Conference on Machine Learning, 2019, pp. 2961–2970.
[138] S. Fan, G. Song, B. Yang, X. Jiang, Prioritized experience replay in multi-actor-attention-critic for reinforcement learning, IOPscience Journal of Physics: Conference Series 1631 (2020). doi:10.1088/1742-6596/1631/1/012040.
[139] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al., IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures, in: PMLR International Conference on Machine Learning, 2018, pp. 1407–1416.
[140] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, M. Michalski, Seed rl: Scalable and efficient deep-rl with accelerated central inference, arXiv:1910.06591 (2019).
[141] H. Hasselt, Double q-learning, Advances in Neural Information Processing Systems 23 (2010) 2613–2621.
[142] Y. Burda, H. Edwards, A. Storkey, O. Klimov, Exploration by random network distillation, arXiv:1810.12894 (2018).
[143] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT Press, 2018.
[144] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Advances in Neural Information Processing Systems, 2000, pp. 1057–1063.
[145] V. R. Konda, J. N. Tsitsiklis, Actor-critic algorithms, in: Advances in neural information processing systems, 2000, pp. 1008–1014.
[146] S.-Y. Shin, Y.-W. Kang, Y.-G. Kim, Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot, MDPI Applied Sciences 9 (24) (2019). doi:10.3390/app9245571.

[147] Q. Liu, D. Wang, Stein variational gradient descent: A general purpose bayesian inference algorithm, arXiv:1608.04471 (2016).
[148] D. Wierstra, A. Förster, J. Peters, J. Schmidhuber, Recurrent policy gradients, Logic Journal of the IGPL 18 (5) (2010) 620–634.
[149] J. Martens, R. Grosse, Optimizing neural networks with kronecker-factored approximate curvature, in: PMLR International conference on machine learning, 2015, pp. 2408–2417.
[150] R. Munos, T. Stepleton, A. Harutyunyan, M. G. Bellemare, Safe and efficient off-policy reinforcement learning, arXiv:1606.02647 (2016).