

Chapter 1

Deep Learning and Reinforcement Learning for Autonomous Unmanned Aerial Systems: Roadmap for Theory to Deployment

Jithin Jagannath, Anu Jagannath, Sean Furman, Tyler Gwin

Abstract Unmanned Aerial Systems (UAS) are being increasingly deployed for commercial, civilian, and military applications. The current UAS state-of-the-art still depends on a remote human controller with robust wireless links to perform several of these applications. The lack of autonomy restricts the domains of application and tasks for which a UAS can be deployed. This is even more relevant in tactical and rescue scenarios where the UAS needs to operate in a harsh operating environment with unreliable wireless links. Enabling autonomy and intelligence to the UAS will help overcome this hurdle and expand its use improving safety and efficiency. The exponential increase in computing resources and the availability of large amount of data in this digital era has led to the resurgence of machine learning from its last winter. Therefore, in this chapter, we discuss how some of the advances in machine learning, specifically deep learning and reinforcement learning can be leveraged to develop next-generation autonomous UAS.

We first begin motivating this chapter by discussing the application, challenges, and opportunities of the current UAS in the introductory section. We then provide an overview of some of the key deep learning and reinforcement learning techniques discussed throughout this chapter. A key area of focus that will be essential to enable autonomy to UAS is computer vision. Accordingly, we discuss how deep learning approaches have been used to accomplish some of the basic tasks that contribute to providing UAS autonomy. Then we discuss how reinforcement learning is explored for using this information to provide autonomous control and navigation for UAS. Next, we provide the reader with directions to choose appropriate simulation suites and hardware platforms that will help to rapidly prototype novel machine learning based solutions for UAS. We additionally discuss the open problems and challenges pertaining to each aspect of developing autonomous UAS solutions to shine light on potential research areas. Finally, we provide a brief account of the UAS safety and regulations prior to concluding the chapter.

Jithin Jagannath, Anu Jagannath, Sean Furman, Tyler Gwin
Marconi-Rosenblatt Innovation Laboratory, ANDRO Computational Solutions, LLC, NY, USA
e-mail: {jjagannath, ajagannath, sfurman, tgwin}@androc.com

1.1 Introduction

The current era of Unmanned Aerial Systems (UASs) has already made a significant contribution to civilian, commercial, and military applications [65, 1]. The ability to have aerial systems perform tasks without having a human operator/pilot in the cockpit has enabled these systems to evolve in different sizes, forms, capabilities, conduct tasks, and missions that were previously hazardous or infeasible. Since the penetration of UAS into different realms of our lives is only going to increase, it is important to understand the current state-of-the-art, determine open challenges, and provide road-maps to overcome these challenges.

The relevance of UASs is increasing exponentially at a Compound Annual Growth Rate (CAGR) of 15.5% to culminate at USD 45.8 billion by 2025 [30]. While this growth seems extremely promising, there are several challenges that need to be overcome before UAS can achieve its full potential. The majority of these UASs are predominantly controlled by an operator and depend on reliable wireless communication links to maintain control and accomplish tasks. As the number of these systems increases and the mission complexity escalates, autonomy will play a crucial role in the next generation of UAS. In the next decade, we will see an incredible push towards autonomy for UAS just like how autonomy has evolved in markets like manufacturing, automotive industry, and in other robotics-related market areas.

When it comes to autonomy, there are several definitions and levels of autonomy claimed by manufacturers. Similarly, several definitions and requirements for various levels of autonomy exist in literature. According to [2], autonomy in UAS can be divided into five levels as follows,

- **Level 1 - Pilot Assistance:** At this initial level, the UAS operator still maintains control of the overall operation and safety of the UAS. Meanwhile, the UAS can take over at least one function (to support navigation or maintaining flight stability) for a limited period of time. Therefore, at this level, the UAS is never in control of both speed and direction of flight simultaneously and all these controls are always with the operator.
- **Level 2 - Partial Automation:** Here, the UAS is capable of taking control of altitude, heading, and speed in some limited scenarios. It is important to understand that the operator is still responsible for the safe operation of the UAS and hence needs to keep monitoring the environment and flight path to take control when needed. This type of automation is predominantly used for application with a pre-planned path and schedules. At this level, the UAS is said to be capable of *sensing*.
- **Level 3 - Conditional Automation:** This case is similar to Level 2 described before with the exception that the UAS can notify the operator using onboard sensors if intervention is needed. This means the operator can be a little more disengaged as compared to Level 2 and acts as the backup controller. It is important to understand that at this level the scenarios of operation are relatively static. If any change in operating conditions is detected, the UAS will alert the operator

to take over the control. At this level, the UAS is said to be capable of *sense and avoid*.

- **Level 4 - High Automation:** At this level, the UAS is designed to operate without the requirement of the controller in several circumstances with the capability to detect and avoid obstacles using several built-in functionalities, rule sets, or machine learning-based algorithms deployed on the embedded computers on the UAS. While the operator can take control of the UAS, it is not necessary since several backup systems are in place to ensure safety in case one system fails. This is where an ideal system is expected to adapt to highly dynamic environments using powerful techniques like machine learning. At this level, the UAS is said to have achieved complete *sense and navigate* capability.
- **Level 5 - Full Automation:** In this final level, the UAS operates fully autonomously without any intervention from operators regardless of the operating scenarios. This will not only include sense and navigate but the ability to learn and adapt its objectives and goals or even optimize its operational objectives and make necessary changes on-the-fly.

Several of today's UASs have limited semi-autonomous modes (level 1 to 3) that warrants UAS to perform some autonomous actions such as return to the initial location, follow a pre-determined flight path, perform maneuvering acts, and recover from some standard instabilities, among others. A completely autonomous system (level 4 and 5) that can interact and survive in a dynamic environment without the need for human-in-the-loop are still far from being realized or deployed in a safe and effective manner.

Machine learning, a subset of Artificial Intelligence has seen a spike in its application in various domains. This resurgence from its last winter is attributed to two main reasons (i) the exponential growth in computing resources in the last decade (ii) digitization of the modern era that has provided access to a huge quantity of data that can be used to train these machine learning models. Today, we see machine learning algorithms successfully applied to computer vision [121, 67, 88], natural language processing [105, 55], medical application [96], wireless communication [76, 75], signal intelligence [77], robotics [109], speech recognition [39], among others. These advancements in the field of machine learning have rendered it a perfect candidate to realize autonomy in UAS. To this end, in this chapter, we discuss the advances made in the field of machine learning, specifically deep learning, and reinforcement learning to facilitate autonomy to UAS. We also look at the key challenges and open research problems that need to be addressed for UAS autonomy. We hope this chapter becomes a great guide to beginners as well as seasoned researchers to take larger strides in these areas of research.

1.1.1 Applications of UAS

The applications of UAS can be broadly divided as follows, (i) Intelligence, Surveillance and Reconnaissance (ISR), (ii) payload/product delivery, and (iii) maintenance

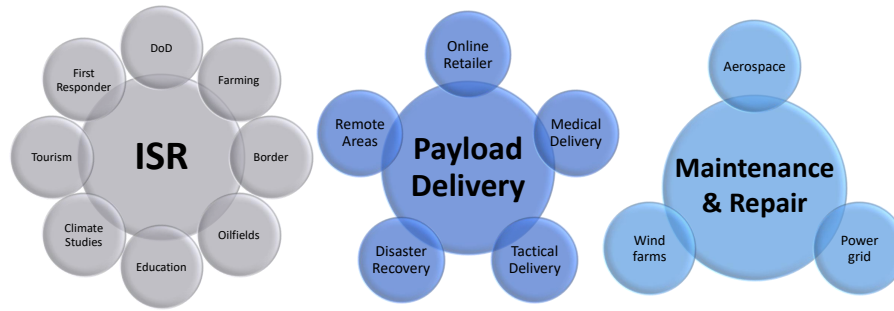


Fig. 1.1 Various applications of UAS

and repair as shown in Figure 1.1. Presently, ISR is the most common application that UASs are employed for in both commercial and military realms. UASs are used for surveillance and remote sensing to map areas of interest using sensors such as traditional cameras or other sensors like acoustic, Infrared (IR), radars, among others. UASs are also used to monitor and survey oilfields, crop surveys, power grids, and other areas that are remote or difficult to access by operators. The surveys are also used for education, environment and climate studies, tourism, mapping, crop assessments, weather, traffic monitoring and border management. Similarly, UASs are used for humanitarian aid and rescue operations by first responders during disasters like flood and earthquakes where access by road does not exist or rendered inaccessible.

UAS is also being actively being designed and developed to become efficient agents for the delivery of payloads. These payloads include packages from online retailers, medical supplies to hospitals or areas of disaster, maintenance parts to remote locations in the commercial and civilian domains. As one can imagine delivery of different kinds of payloads will also be critical for several military missions and UAS might provide a safer alternative to accomplish such delivery in hostile areas with limited accessibility. Though not prevalent yet, it is envisioned that UAS will open up the market for several maintenance and repair tasks for the aerospace industry, power grid, wind farms, and other operations that are not easy to access. Currently, UAS are already being deployed to monitor and detect faults as well as provide maintenance alerts to reduce operational expense. It is envisioned that robotics enabled UAS will also be able to intervene when faults or necessary repair are detected in the near future.

1.1.2 Classification of UAS

It is clear from the previous discussion about the applications of UAS the need for versatility in the UAS design. Due to these reasons and the immense prospective that UAS holds for the future, UASs have evolved into different forms and sizes.

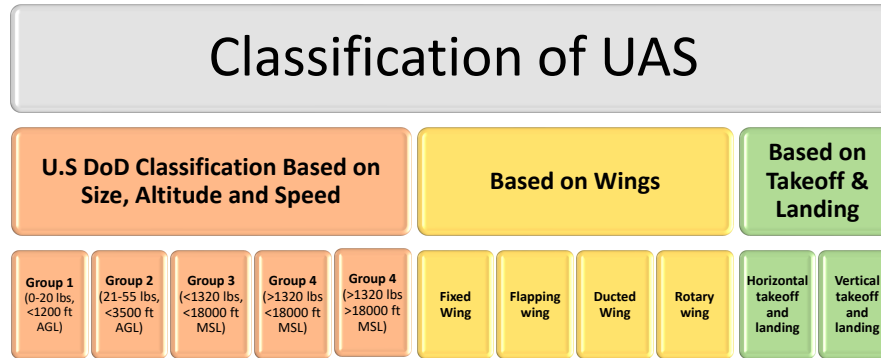


Fig. 1.2 Classification of Unmanned Aerial Systems

While most of the discussion in this chapter is not specific to any type of UAS platforms, we provide a succinct classification for UASs in Figure 1.2. Several characteristics are used to classify different types of UASs. Here, we present the three most prevalent ones. The first is the classification of UASs adopted by Department of Defense (DoD) which divides the systems into five groups based on weight, the altitude of flight, and the velocity. Another two sets of classification are based on the wing type and landing and takeoff. All these have been summarized in Figure 1.2.

1.1.3 Chapter Organization

This chapter is written for the benefit of a broad array of readers who have different levels of understanding and experience in this area of research. Therefore, for the benefit of readers who are relatively new to machine learning, we start by providing an overview of specific machine learning techniques that are explored in this chapter. The detailed explanation of these techniques would ensure even a beginner in the area of machine learning to grasp these techniques and benefit from the rest of the discussion in the chapter. The core contribution of this chapter is presented in the next four sections. Among these, two sections are dedicated to the discussion of various deep learning and reinforcement learning that has been explored for UAS. In each of these sections, we also discuss the open problems and challenges to motivate researches to explore these areas further. Since the goal of every research endeavor is to ensure the novel algorithms and solutions are effectively deployed on target platforms, in the next two sections, we look at simulation suites and hardware platforms that can help expedite this process. Finally, we conclude the chapter in the final section. The overall chapter organization is depicted in Figure 1.3.

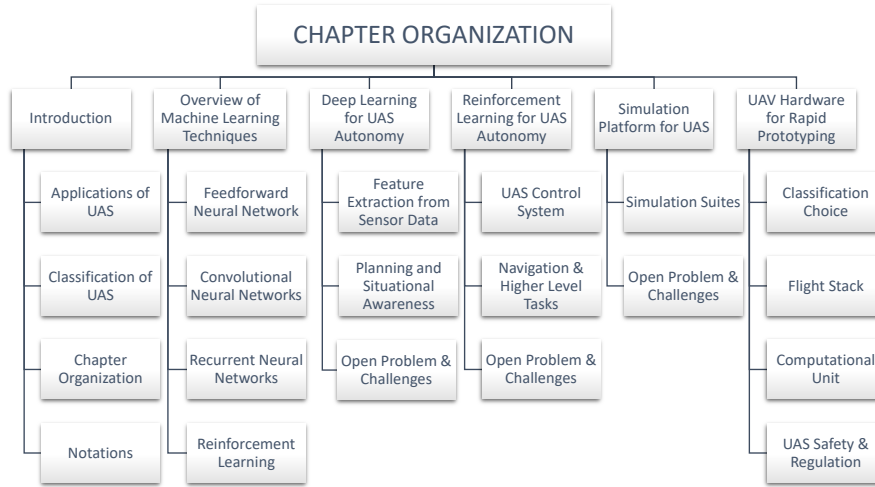


Fig. 1.3 Organization of the Chapter

1.1.4 Notations

Here, we introduce some standard notations that will be used throughout this chapter. Matrices and vectors will be denoted by boldface upper and lower-case letters, respectively. For a vector \mathbf{x} , x_i denotes the i -th element, $\|\mathbf{x}\|$ indicates the Euclidean norm, \mathbf{x}^\top represents its transpose, and $\mathbf{x} \cdot \mathbf{y}$ the Euclidean inner product of \mathbf{x} and \mathbf{y} . For a matrix \mathbf{H} , H_{ij} will indicate the element at row i and column j . The notation \mathbb{R} and \mathbb{C} will indicate the set of real and complex numbers, respectively. The notation $\mathbb{E}_{x \sim p(x)} [f(x)]$ is used to denote the expected value, or average of the function $f(x)$ where the random variable x is drawn from the distribution $p(x)$. When a probability distribution of a random variable, x , is conditioned on a set of parameters, θ , we write $p(x; \theta)$ to emphasize the fact that θ parameterizes the distribution and reserve the typical conditional distribution notation, $p(x|y)$, for the distribution of the random variable x conditioned on the random variable y . We use the standard notation for operations on sets where \cup and \cap are the infix operators denoting the union and intersection of two sets, respectively. We use $S_k \subseteq S$ to say that S_k is either a strict subset of or equal to the set S and $x \in S$ to denote that x is an element of the set S . \emptyset is used to denote the empty set and $|S|$ represents the cardinality of a set S . Lastly, the convolution operator is denoted as $*$.

1.2 Overview of Machine Learning Techniques

Machine Learning is a branch of artificial intelligence that is able to learn patterns from raw data and/or learn from observation sampling from the environment en-

abling computer systems to acquire knowledge. Machine learning is broadly classified into supervised, unsupervised, and reinforcement learning which are further subdivided into subcategories as shown in Fig.1.4 (this is a very limited/relevant representation of this vast field). In this section, we elaborate on the key machine learning techniques (which are indicated as gray boxes in the Fig.1.4) prominently used in this chapter to benefit readers in understanding the deep learning approaches for UAS autonomy.

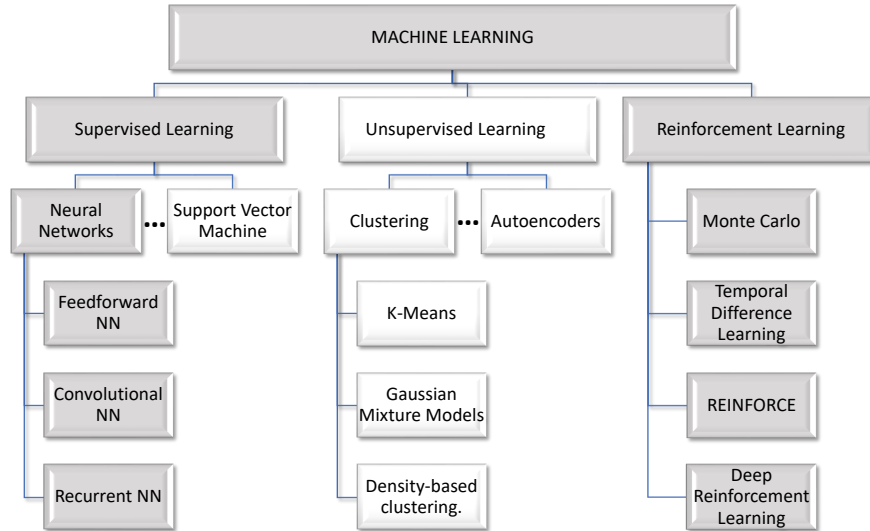


Fig. 1.4 Machine Learning Techniques

1.2.1 Feedforward Neural Networks

Feedforward neural networks (FNN) also referred to as multilayer perceptrons are directed layered neural networks with no internal feedback connections. Mathematically, an FNN performs a mapping, i.e., $f : X \rightarrow Y$. An N-layered FNN is a composite function $y = f(\mathbf{x}; \theta) = f_N(f_{N-1}(\dots f_1(\mathbf{x})))$ mapping input vector $\mathbf{x} \in \mathbb{R}^m$ to a scalar output $y \in \mathbb{R}$. Here, θ represents the neural network parameters. The number of layers in the neural network dictates the *depth* whereas the number of neurons in the layers defines the *width* of the network. The layers in between the input and output layers for which the output does not show are called *hidden* layers. Figure 1.5 shows a 3-layered FNN accepting a two-dimensional input vector $\mathbf{x} \in \mathbb{R}^2$ approximating it to a scalar output $y \in \mathbb{R}$.

In the figure, each node represents a neuron and each link between the nodes i and j are assigned a weight w_{ij} . The composite function of the 3-layered FNN is

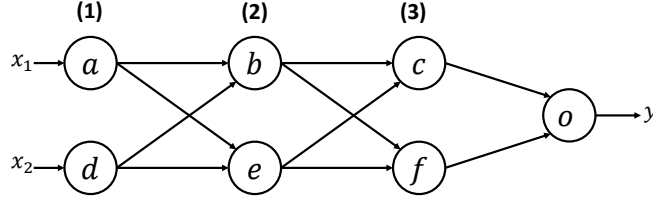


Fig. 1.5 Three-layered FNN

$$y = f(\mathbf{x}; \boldsymbol{\theta}) = f_3(f_2(f_1(\mathbf{x}))) \quad (1.1)$$

In other words, the 3-layer FNN in Fig.1.5 is the directed acyclic graph equivalent of the composite function in equation (1.1). The mapping in the first layer is

$$\mathbf{h}_1 = f_1(\mathbf{x}) = \mathcal{A}_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \quad (1.2)$$

where $\mathcal{A}_1(\circ)$ is the activation function, \mathbf{b}_1 is the bias vector, and \mathbf{W}_1 represents the weight matrix between the neurons in the first and second layers. Here, the weight matrix \mathbf{W}_1 is defined as the link weights between the neurons in the input and second layer

$$\mathbf{W}_1 = \begin{bmatrix} w_{ab} & w_{db} \\ w_{ae} & w_{de} \end{bmatrix}. \quad (1.3)$$

Similarly, the second layer mapping can be represented as

$$\mathbf{h}_2 = f_2(\mathbf{h}_1) = \mathcal{A}_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) \quad (1.4)$$

Finally, the output is

$$y = f_3(\mathbf{h}_2) = \mathcal{A}_3(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3) \quad (1.5)$$

The weight matrices in the second and final layers are

$$\mathbf{W}_2 = \begin{bmatrix} w_{bc} & w_{ec} \\ w_{bf} & w_{ef} \end{bmatrix} \text{ and } \mathbf{W}_3 = [w_{co} \ w_{fo}].$$

The neural network parameters $\boldsymbol{\theta} = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$ comprise the weight matrices and bias vectors across the layers. The objective of the training algorithm is to learn the optimal $\boldsymbol{\theta}^*$ to get the target composite function f^* from the available samples of \mathbf{x} .

1.2.2 Convolutional Neural Networks

Convolutional networks or convolutional neural networks (CNNs) are a specialized type of feedforward neural network that performs convolution operation in at least

one of its layers. The *feature extraction* capability of CNNs mimics the neural activity of the animal visual cortex [89]. The visual cortex comprises a complex arrangement of cells that are sensitive to sub-regions of the perceived scene. The convolution operation in CNNs emulates this characteristic of the brain's visual cortex. Consequently, CNNs have been abundantly applied in the field of computer vision [129, 88, 91, 125, 73, 92, 113, 103, 68]. The convolution is an efficient method of feature extraction that reduces the data dimension and consequently reduces the parameters of the network. Hence, CNNs are more efficient and easier to train in contrast to its fully connected feedforward counterpart 1.2.1.

A typical CNN architecture would often involve convolution, pooling, and output layers. CNNs operate on input tensor $\mathbf{X} \in \mathbb{R}^{W \times H \times D}$ of width W , height H , and depth D which will be operated on by kernel (filter) $\mathbf{K} \in \mathbb{R}^{w \times h \times D}$ of width w , height h , and of the same depth as the input tensor to generate an output feature map $\mathbf{M} \in \mathbb{R}^{W_1 \times H_1 \times D_1}$. The dimension of the feature map is a function of the input as well as kernel dimensions, the number of kernels N , stride S , and the amount of zero padding P . Likewise, the feature map dimensions can be derived as $W_1 = (W - w + 2P)/S + 1$, $H_1 = (H - h + 2P)/S + 1$, $D_1 = N$. Each kernel slice extracts a specific feature from the input region of operation. Kernel refers to the set of weights and biases. The kernel operates on the input slice in a sliding window manner based on the stride. Stride refers to the number of steps with which to slide the kernel along with the input slice. Hence, each depth slice of the input is treated with the same kernel or in other words, shares the same weights and biases - *parameter sharing*. The convolution operation on an input slice \mathbf{x} by a kernel \mathbf{k} is demonstrated in Fig.1.6. Here, b represents the bias associated with the kernel slice and $\mathcal{A}(\circ)$ denotes a non-linear activation function.

Input Slice			*	Kernel		=	Feature Map	
x_{11}	x_{12}	x_{13}	*	k_{11}	k_{12}	=	$m_{11} = \mathcal{A}(x_{11}k_{11} + x_{12}k_{12} + x_{21}k_{21} + x_{22}k_{22} + b)$	$m_{12} = \mathcal{A}(x_{12}k_{11} + x_{13}k_{12} + x_{22}k_{21} + x_{23}k_{22} + b)$
x_{21}	x_{22}	x_{23}		k_{21}	k_{22}		$m_{21} = \mathcal{A}(x_{21}k_{11} + x_{22}k_{12} + x_{31}k_{21} + x_{32}k_{22} + b)$	$m_{22} = \mathcal{A}(x_{22}k_{11} + x_{23}k_{12} + x_{32}k_{21} + x_{33}k_{22} + b)$
x_{31}	x_{32}	x_{33}						

Fig. 1.6 Convolution of input slice with kernel

The resulting output from the convolution operation is referred to as the *feature map*. Each element of the feature map can be visualized as the output of a neuron which focuses on a small region of the input - *receptive field*. The neural depiction of the convolution interaction is shown in Fig.1.7.

It is evident that each neuron in a layer is connected locally to the neurons in the adjacent layer - *sparse connectivity*. Hence, each neuron is unaffected by variations outside of its receptive field while producing the strongest response for spatially local input pattern. The feature maps are propagated to subsequent layers until it reaches the output layer for a regression or classification task. *Pooling* is a typical operation in CNN to significantly reduce the dimensionality. It operates on a subre-

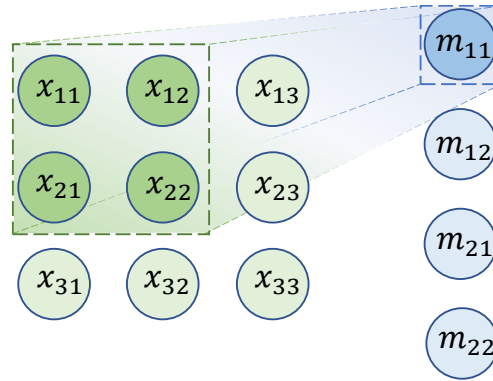


Fig. 1.7 Neural representation of convolution

gion of the input to map it to a single summary statistic depending on the type of pooling operation - max, mean, L_2 -norm, weighted average, etc. In this way, pooling downsamples its input. A typical pooling dimension is 2×2 . Larger pooling dimensions might risk losing significant information. Figure 1.8 shows max and mean pooling operations.

Input Slice				Pooled Output	
5	3	1	2×2 Max Pool	10	14
2	10	14		10	14
2	2	6			

Input Slice				Pooled Output	
5	3	1	2×2 Mean Pool	5	7
2	10	14		4	8
2	2	6			

Fig. 1.8 Max and mean pooling on input slice with stride 1

A pooling layer of dimensions $W_p \times H_p$ upon operating over an input volume of size $W_1 \times H_1 \times D_1$ with a stride of S_1 will yield an output of volume $W_2 = (W_1 - W_p) / S_1$, $H_2 = (H_1 - H_p) / S_1$, $D_2 = D_1$. Pooling imparts invariance to translation, i.e., if the input to the pooling layer is shifted by a small amount, the pooled output will largely be unaffected [64].

As we have discussed, the three essential characteristics of CNNs that contribute to the statistical efficiency and trainability are parameter sharing, sparse connectivity, and dimensionality reduction. CNNs have demonstrated superior performance in computer vision tasks such as image classification, object detection, semantic

scene classification, etc. Consequently, CNNs are increasingly used for UAS imagery and navigation applications [51]. Most notable CNN architectures are LeNet-5 [91], AlexNet [88], VGG-16 [125], ResNet [68], Inception [129], and SqueezeNet [73].

1.2.3 Recurrent Neural Networks

Recurrent Neural Network (RNN) [115] is a type of feedforward neural network specialized to capture temporal dependencies from sequential data. RNN holds internal memory states and recurrent connections between them to capture the sequence history. This characteristic of RNN enables it to exploit the temporal correlation of data rendering them suitable for image captioning, video processing, speech recognition, and natural language processing applications. Unlike CNN and traditional feedforward neural networks, RNN can handle variable-length input sequences with the same model.

RNNs operate on input sequence vectors at varying time steps \mathbf{x}^t and map it to output sequence vectors \mathbf{y}^t . The recurrence relation in an RNN parameterized by θ can be expressed as

$$\mathbf{h}^t = \mathcal{F}(\mathbf{h}^{t-1}, \mathbf{x}^t; \theta) \quad (1.6)$$

where \mathbf{h}^t represents the hidden state vector at time t . The recurrence relation represents a recursive dynamic system. By this comparison, RNN can be defined as *a recursive dynamic system that is driven by an external signal, i.e. input sequence \mathbf{x}^t* . The equation (1.6) can be unfolded twice as

$$\mathbf{h}^t = \mathcal{F}(\mathbf{h}^{t-1}, \mathbf{x}^t; \theta) \quad (1.7)$$

$$= \mathcal{F}(\mathcal{F}(\mathbf{h}^{t-2}, \mathbf{x}^{t-1}; \theta), \mathbf{x}^t; \theta) \quad (1.8)$$

$$= \mathcal{F}(\mathcal{F}(\mathcal{F}(\mathbf{h}^{t-3}, \mathbf{x}^{t-2}; \theta), \mathbf{x}^{t-1}; \theta), \mathbf{x}^t; \theta) \quad (1.9)$$

The unfolded equations show how RNN processes the whole past sequences $\mathbf{x}^t, \mathbf{x}^{t-1}, \dots, \mathbf{x}^1$ to produce the current hidden state \mathbf{h}^t . Another notable inference from the unfolded representation is the *parameter sharing*. Unlike CNN, where the parameters of a spatial locality are shared, in an RNN, the parameters are shared across different positions in time. For this reason, RNN can operate on variable-length sequences allowing the model to learn and generalize well to inputs of varying forms. On the other hand, traditional feedforward network does not share parameters and have a specific parameter per input feature preventing it from generalizing to an input form not seen during training. At the same time, CNN share parameter across a small spatial location but would not generalize to variable-length inputs as well as an RNN. A simple many-to-many RNN architecture which maps multiple input sequences to multiple output sequences is shown in Fig.1.9.

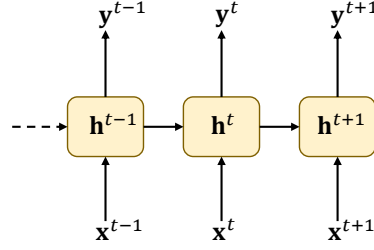


Fig. 1.9 Many-to-many RNN architecture

For a simple representation, let us assume the RNN is parameterized by θ and ϕ with input-to-hidden, hidden-to-hidden, and hidden-to-output weight matrices being \mathbf{W}_{ih} , \mathbf{W}_{hh} , and \mathbf{W}_{ho} respectively. The hidden state at time t can be expressed as

$$\mathbf{h}^t = \mathcal{F}(\mathbf{h}^{t-1}, \mathbf{x}^t; \theta) \quad (1.10)$$

$$= \mathcal{A}_h(\mathbf{W}_{hh}\mathbf{h}^{t-1} + \mathbf{W}_{ih}\mathbf{x}^t + \mathbf{b}_h). \quad (1.11)$$

where $\mathcal{A}_h(\circ)$ is the activation function of the hidden unit and \mathbf{b}_h is the bias vector. The output at time t can be obtained as a function of the hidden state at time t ,

$$\mathbf{y}^t = \mathcal{G}(\mathbf{h}^t; \phi) \quad (1.12)$$

$$= \mathcal{A}_o(\mathbf{W}_{ho}\mathbf{h}^t + \mathbf{b}_o) \quad (1.13)$$

where $\mathcal{A}_o(\circ)$ is the activation function of the output unit and \mathbf{b}_o is the bias vector. Other typical RNN architectures are shown in Fig.1.10.

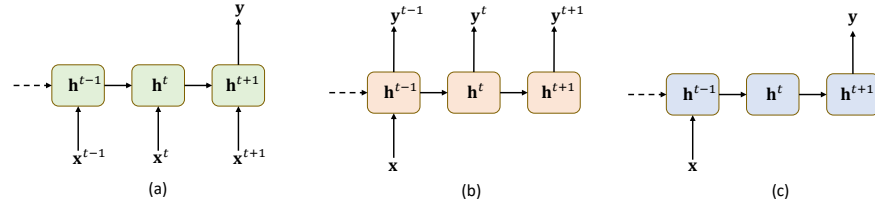


Fig. 1.10 RNN architectures. (a) Many-to-one, (b) One-to-many, and (c) One-to-one

The RNN architectures discussed so far captures only hidden states from the past. Some applications would also require future states in addition to past. This is accomplished by a bidirectional RNN [120]. In simple words, bidirectional RNN combines an RNN that depends on past states (*i.e.*, from $\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3, \dots, \mathbf{h}^t$) with that of an RNN which looks at future states (*i.e.*, from $\mathbf{h}^t, \mathbf{h}^{t-1}, \mathbf{h}^{t-2}, \dots, \mathbf{h}^1$).

1.2.4 Reinforcement Learning

Reinforcement learning is focused on the idea of a goal-directed agent interacting with an environment based on its observations of the environment [128]. The main goal of reinforcement learning is for the agent to learn how to act i.e., what action to perform in a given environmental state, such that a reward signal is maximized. The agent repeatedly interacts with the environment in a series of discrete time steps by observing the environmental state, choosing, and executing an action. The action chosen by the agent may affect the state of the environment in the next time step. The agent receives a reward signal from the environment and transitions to a new state. The agent has some capability to sense the environmental state; informally the state can be thought of as any information about the environment that is made available to the agent. The agent selects which of the possible actions it can take by following a policy which is a function, in general stochastic, that maps state to actions. A reward signal is used to define the goal of the problem. The reward received by the agent at each time step specifies the immediate desirability of the current state. The objective of the reinforcement learning agent is to maximize the cumulative reward, typically defined by a value function, which defines the long-term goodness of the agent. The agent aims at achieving a goal by continuously interacting with the environment. This interaction which involves taking actions while trading off short and long term rewards renders reinforcement learning a potentially well-suited solution to many autonomous problems [83].

The reinforcement learning problem is usually represented mathematically using a finite Markov Decision Process (MDP). A finite MDP is defined by the following tuple (S, A, P, R) , where S , A , P , and R are the state space, action space, transition function, and reward function respectively. Note that in finite MDPs, the state, action, and reward spaces consist of a finite number of elements. At each time step, the agent observes state $s \in S$, selects and takes action $a \in A$, receives a reward r , and transitions to a new state $s' \in S$. The transition function specifies the probability of transitioning from state s to state s' as a consequence of choosing action a as,

$$P(s, a, s') = Pr(S_{t+1} = s' | S_t = s, A_t = a). \quad (1.14)$$

The reward function R defines the expected reward received by the agent after transitioning to state s' from state s after taking action a i.e.,

$$R(s, a) = \mathbb{E}[R_t | S_t = s, A_t = a]. \quad (1.15)$$

It can be seen that the functions P and R define the dynamics of the MDP. A reinforcement learning agent uses a policy to select actions in a given state. The policy, denoted $\pi(s, a)$ provides a probabilistic mapping of states to actions as,

$$\pi(s, a) = Pr(A_t = a | S_t = s). \quad (1.16)$$

As discussed earlier, value functions are used to define the long term goodness of the agent. Mathematically, the *state-value function* is denoted as

$$v_{\pi}(s) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \forall s \in S \quad (1.17)$$

The *state-value function* specifies the expected return, i.e., sum of discounted rewards, if the agent follows policy π starting from state s . The discount rate γ , $0 \leq \gamma \leq 1$, is used to weight future rewards progressively less. For example, as γ approaches zero the agent is concerned only with immediate rewards whereas when γ approaches unity, the agent favors future rewards. The expected discounted return is denoted by G_t i.e.,

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \quad (1.18)$$

Additionally, the *action-value function* for policy π is mathematically represented as

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \quad (1.19)$$

The *action-value function* specifies the expected return if the agent takes action a in state s under policy π .

The MDP dynamics of the environment and the notion of value functions have been exploited to develop multiple algorithms. In the case where the MDP is fully known, i.e., the agent has knowledge of P and R , dynamic programming methods (planning algorithms), such as policy iteration and value iteration can be used to solve the MDP for the optimal policy or optimal value function. However, in reinforcement learning, knowledge of the MDP dynamics is not usually assumed. Both model-based and model-free approaches exist for solving reinforcement learning problems. In model-based reinforcement learning, the agent attempts to learn a model of the environment directly, by learning P and R , and then using the environmental model to plan actions using algorithms similar to policy iteration and value iteration. In model-free reinforcement learning, the agent does not attempt to directly learn a model of the environment but rather attempts to learn an optimal value function or policy. The discussion in this chapter is primarily focused on model-free methods.

Generally speaking, model-free reinforcement learning algorithms fall into value function or policy gradient based methods. In value function based methods, the agent attempts to learn an optimal value function, usually action-value, and from which an optimal policy can be found. Value function methods include Monte Carlo, State-action-reward-state-action (SARSA), and Q-Learning. Policy gradient based methods attempt to learn an optimal parameterized policy directly via a gradient of a scalar performance measure with respect to the policy parameter. The REINFORCE algorithm is an example of a policy gradient method.

Monte Carlo

Monte Carlo methods can be utilized to learn value functions and optimal policies by direct experience with the environment. In particular, sequences of states, actions, and rewards can be obtained by the agent interacting with the environment, either directly or in simulation, and the value function can be estimated by averaging the returns beginning from a state-action pair. Monte Carlo methods are typically used for episodic tasks. An episode (sequence of state, action, reward) is generated by the agent following policy π in the environment and the value function estimate is updated at the conclusion of each episode. Monte Carlo methods can be used for control i.e., finding the optimal policy, by performing policy improvement. Policy improvement updates the policy such that it is greedy with respect to the current action-value function estimate. The greedy policy for an action-value function is defined such that for each state s the action with the maximum action-value is taken i.e.,

$$\pi(s) \doteq \operatorname{argmax}_{a \in A} q(s, a). \quad (1.20)$$

An important consideration for using Monte Carlo methods for value function prediction, and in reinforcement learning in general, is that of maintaining exploration. In order to learn the action-value function, all state-action pairs need to be explored. One way to achieve this is known as exploration whereby each episode begins in a particular state-action pair and all state-action pairs have a non-zero probability of being selected at the start of an episode. Exploration guarantees every state-action pairs will be visited an infinite number of times in the limit of an infinite number of episodes [128]. An alternative approach is to utilize a policy that allows for continued exploration. An example is the ε -greedy policy in which most of the time an action (probability of $1 - \varepsilon$) is selected that maximizes the action-value function while occasionally a random action is chosen with probability ε i.e.,

$$\pi(s, a) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|}, & \text{if } a = a^* \\ \frac{\varepsilon}{|A|}, & \text{otherwise} \end{cases} \quad (1.21)$$

There are two approaches to ensure continued exploration: on-policy and off-policy methods. In on-policy methods, the algorithm attempts to evaluate and improve the policy that is being used to select actions in the environment whereas off-policy methods are improving a policy different than the policy used to select actions. In off-policy methods, the agent attempts to learn an optimal policy, called the target policy, by generating actions using another policy that allows for exploration, called the behavior policy. Since the policy learning is from data collected “off” the target policy, the methods are called off-policy. Both on-policy and off-policy Monte Carlo control methods exist.

Temporal Difference Learning

Temporal Difference (TD) learning defines another family of value function based reinforcement learning methods. Similar, to Monte Carlo methods, TD learns a value function via interaction with the environment. The main difference between TD and Monte Carlo is that TD updates its estimate of the value function at each time step rather than at the end of the episode. In other words, the value function update is based on the value function estimate of the subsequent state. The idea of updating value function based on the estimated return ($R_{t+1} + \gamma V(S_{t+1})$) rather than the actual (complete) reward as in Monte Carlo is known as bootstrapping. A simple TD update equation for value function is

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (1.22)$$

where α is a step size parameter. In the above equation, it is seen that the TD method updates the value function estimate at the next time step. The target value for the TD update becomes $R_{t+1} + \gamma V(S_{t+1})$ which is compared to the current value function estimate ($V(S_t)$). The difference between the target and the current estimate is known as the TD error i.e.,

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \quad (1.23)$$

It can be seen that an advantage of TD methods is its ability to update value function predictions at each time step which enables online learning.

SARSA: is an example of an on-policy TD control algorithm. The TD update equation presented above is extended for action-value function prediction yielding the SARSA action-value update rule as,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \quad (1.24)$$

As shown in the equation (1.24), the update is performed after each sequence of $(\dots, S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \dots)$ which leads to the name SARSA. It is to be noted that the Q estimate is updated based on the sample data generated from the behavior policy ($R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$). For the control algorithm perspective, a greedy policy like ϵ -greedy is often used.

Q-Learning: is an off-policy TD control algorithm and its update rule is given below.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (1.25)$$

As an off-policy method, the learned action-value function estimate Q is attempting to approximate the optimal action-value function Q^* directly. This can be seen in the update equation (1.25) where the target value is $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ compared to $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ of SARSA. Unlike in SARSA, the Q value is updated based on the greedy policy for action selection rather than the behavior policy. SARSA does not learn the optimal policy but rather learns the action-values resulting from the ϵ -greedy action selections. However, Q-learning learns the optimal policy re-

sulting from the ϵ -greedy action selections causing the online performance to drop occasionally [128].

The TD methods can be further generalized with n -step bootstrapping methods which are an intermediate between Monte Carlo and TD approaches. The n -step methods generalize the TD methods discussed earlier by utilizing the next n rewards, states, and actions in the value or action-value function updates.

The value function based approaches discussed so far have been presented as tabular methods. The algorithms are tabular because the state-value or action-value function is represented as a table or an array. In many practical problems of interest, the state spaces are very large and it becomes intractable to learn optimal policies using tabular methods due to the time, data, and memory requirements to populate the tables. Additionally with massive state spaces, it is typical that the agent will enter states that are previously unseen requiring the agent to generalize from experiences in similar states. An example of an overwhelmingly large state space occurs when the environmental state is represented as a camera image; for example, an 8-bit, 200x200 pixel RGB image results in $256^{3*200*200}$ possible states. To cope with these challenges, optimal policies can be approximated by utilizing function approximation techniques to represent value functions and policies. The different function approximation techniques used in supervised learning can be applied to reinforcement learning. The specific use of deep neural networks as a means for function approximation is known as Deep Reinforcement Learning (DRL) and is discussed later in this section.

When using function approximation techniques, parameterized state-value or action-value functions are used to approximate value functions. A state-value estimate can be denoted as $\hat{v}(s; w) \approx v_\pi(s)$ and an action-value estimate as $\hat{q}(s, a; w) \approx q_\pi(s, a)$ where $w \in \mathbb{R}^d$ is the parameter vector. In principle, any supervised learning method could be used for function approximation. For example, a value function estimate could be computed using techniques ranging from a linear function of the state and weights to nonlinear methods such as an Artificial Neural Network (ANN). Stochastic Gradient Descent (SGD) and its variants are often used to learn the value of the parameter vectors.

REINFORCE

In contrast to value function based approaches, policy gradient methods attempt to learn an optimal parameterized policy directly without the requirement of learning the action-value function explicitly. The policy that is learned is defined as

$$\pi(a|s, \theta) = Pr(A_t = a | S_t = s, \theta_t = \theta) \quad (1.26)$$

which specifies the probability that action a is taken at step t in state s and is parameterized by the vector $\theta \in \mathbb{R}^m$. Policy gradient methods learn the value of the policy parameter based on the gradient of a performance measure $J(\theta)$ with respect to the parameter. In the episodic case, the performance measure can be defined in terms of

the state value function assuming the episode starts from an initial state s_0 as

$$J(\theta) \doteq v_{\pi_\theta}(s_0) \quad (1.27)$$

REINFORCE is an example of a policy gradient algorithm and is derived from the policy gradient theorem

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla_\theta \pi(a|s, \theta) \quad (1.28)$$

where $\mu(s)$ is a distribution over states and the gradients are column vectors with respect to parameter vector θ . The policy gradient theorem provides an expression for the gradient of the performance measure with respect to the parameter vector. From the policy gradient theorem, the following equation is derived for the gradient of $J(\theta)$

$$\nabla J(\theta) = \mathbb{E}_\pi \left[G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \quad (1.29)$$

Using SGD, the REINFORCE update rule for the policy parameter vector θ can be derived as

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)}. \quad (1.30)$$

The update equation (1.30) moves the parameter in a direction that increases the probability of taking action A_t during future visits to the state S_t in proportion to the return G_t . This causes the parameter to favor actions that produce the highest return. The normalization prevents choosing actions with a higher probability that may not actually produce the highest return.

It is possible to generalize the policy gradient theorem and REINFORCE update rule with the addition of a baseline for comparison to the action values or returns. The baseline can be an arbitrary function or random variable. The motivation behind the use of a baseline is to reduce the variance in policy parameter updates. The update rule for the REINFORCE algorithm with a baseline is given as

$$\theta_{t+1} = \theta_t + \alpha (G_t - b(S_t)) \frac{\nabla_\theta \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \quad (1.31)$$

where $b(S_t)$ is the baseline. A common baseline is an estimate of the state-value $\hat{v}(S_t, \mathbf{w})$ parameterized by the weight vector $\mathbf{w} \in \mathbb{R}^l$. The idea of using a state-value function as a baseline can be extended with actor-critic methods. In actor-critic methods, a state-value function, called a critic, is utilized to assess the performance of a policy, called an actor. The critic introduces a bias to the actor's gradient estimates which can substantially reduce variance.

The two most recent policy gradient methods are Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO). TRPO was introduced in [118] in order to prevent drastic policy changes by introducing an optimization constraint - Kullback-Leibler (KL) divergence. The policy is updated based on a

trust-region and the KL constraint ensures that the policy update is not too far away from the original policy. The inclusion of KL constraint in the optimization problem introduces computational and implementation difficulty. However, PPO introduced in [119] mitigates this implementation hurdle by incorporating the constraint term within the objective function. PPO computes the probability ratio between new and old policies. There are two variants of PPO - PPO with KL penalty and PPO with clipped objective. In the first variant, the KL constraint is introduced as a penalty term in the objective function such that it computes a policy update that does not deviate much from the previous policy while minimizing the cost function. In the second variant, the KL divergence is replaced with a clipped objective function such that the advantage function will be clipped if the probability ratio lies outside a range, say $1 \pm \phi$. In contrast to TRPO, PPO is simpler to implement and tune.

Deep Reinforcement Learning

Deep Reinforcement Learning is a popular area of current research that combines techniques from deep learning and reinforcement learning [42]. In particular, deep neural networks are used as function approximators to represent action-value functions and policies used in traditional reinforcement learning algorithms. This is of particular interest for problems that involve large state and action spaces that become intractable to represent using tabular methods or traditional supervised learning function approximators. A key capability of deep learning architectures is the ability to automatically learn representations (features) from raw data. For example, a deep neural network trained for image classification will automatically learn to recognize features such as edges, corners, etc. The use of deep learning enables policies to be learned in an end-to-end fashion, for example, learning control policies directly from raw sensor values. A famous exemplary deep reinforcement learning algorithm is the deep Q-Network that pairs Q-Learning with a deep Convolutional Neural Network (CNN) to represent the action-value function [101]. The deep Q-Network was able to achieve super human performance on several Atari games by using only visual information, reward signal, and available actions i.e., no game specific information was given to the agent. The deep Q-Network employs two methods to address the known convergence issues [130] that can arise when using neural networks to approximate the Q function. These methods are experience replay and the use of a separate target network for Q updates. The experience replay mechanism stores sequences of past experiences, $(s_t, a_t, s_{t+1}, r_{t+1})$, over many episodes in replay memory. The past experiences are used in subsequent Q function updates which improve data efficiency, removes correlations between samples, and reduces the variance of updates. The separate target network \hat{Q} is used for generating targets in the Q-Learning updates. The target network is updated every C time steps as a clone of the current Q network; the use of the target network reduces the chances of oscillations and divergence. A variation of the deep Q-network, known as a Deep Recurrent Q-Network [66], adds a Long Short Term Memory (LSTM) layer to help learn temporal patterns. Additional variations include the double deep Q-network, and Du-

eling Double Deep Q-Network (D3QN). Furthermore, deep reinforcement learning has also been applied to problems with continuous action spaces. In [94], an actor-critic algorithm known as Deep Deterministic Policy Gradient (DDPG) is presented that is based on the Deterministic Policy Gradient (DPG) algorithm which exploits the idea of experience replay and target networks from the Deep Q-Network (DQN) as well as batch normalization. DDPG is applied successfully to many continuous control problems. In [69] Recurrent Deterministic Policy Gradient (RDPG) is introduced as an extension to DDPG by the addition of recurrent LSTM. The characteristics and capabilities of deep reinforcement learning warrant further investigation for its application to autonomous Unmanned Aerial Vehicle (UAV) applications.

A summary of the different model-free reinforcement learning algorithms is shown in Figure 1.11.

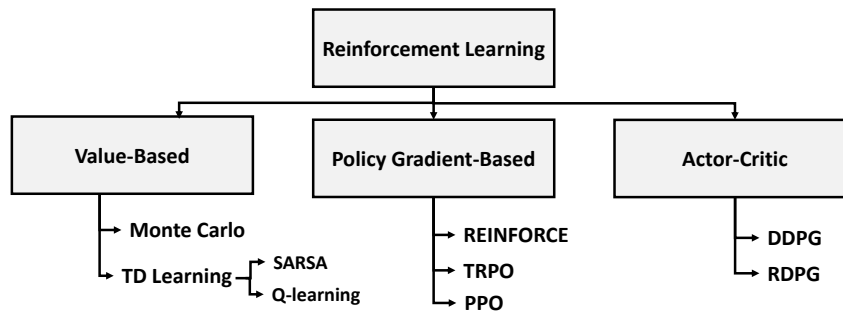


Fig. 1.11 Model-free reinforcement learning algorithms

1.3 Deep Learning for UAS Autonomy

Deep learning has shown great potential in learning complex representations from real environmental data. Its excellent learning capability has shown outstanding results in solving autonomous robotic tasks such as gait analysis, scene perception, navigation, etc., [136, 137]. The same aspects can be applied for enabling autonomy to the UAS. The various UAS focus areas where deep learning can be applied are scene perception, navigation, obstacle and collision avoidance, swarm operation, and situational awareness. This is also exemplified in the Fig.1.12.

Deep learning has been applied as a feature extraction system to learn a high dimensional data representation from the raw sensor output. On the other hand, planning and situational awareness, involve several sub-tasks such as querying or surveying aerial images, navigation control/guidance, collision avoidance, position-dependent control actions, etc. Accordingly, we classify this section into two broad categories: (i) Feature extraction from sensor data and (ii) UAS path planning and situational awareness.

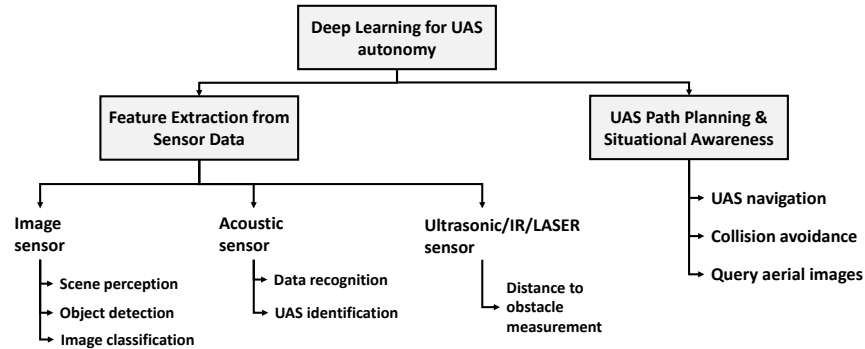


Fig. 1.12 Deep learning for UAS autonomy discussed in this section.

1.3.1 Feature Extraction from Sensor Data

The authors of [63] demonstrated the accuracy of a supervised deep learning image classifier to process the monocular images. The classifier predicted outputs of the forest trail direction such as left, right, or straight and claims an accuracy comparable to humans tested on the same image classification task. This scene perception task will require the Micro Aerial Vehicle (MAV) to perceive the trail and react (take actions) to stay on the trail. The authors adopted a typical CNN architecture to accomplish the supervised image classification task. The CNN involved four convolutional layers interlaced with max pooling layers and concluding with two fully connected layers. The output fully connected layer adopted softmax classification that yields the probability of the input image to belong to a particular class. The network was trained using SGD. The direction estimates from the CNN were extended to provide navigation control. The navigation control for autonomous trail following was tested on ParrotAR Drone interfaced with a laptop and a standalone quadrotor. The paper reported lower classification accuracy for the real-world testing conditions as opposed to the good quality GoPro images in the training dataset.

The AlexNet [88] architecture was employed for palm tree detection and counting in [93] from aerial images. The images were collected from the QuickBird satellite. A sliding window technique with a window size of 17×17 pixels and a stride of 3 pixels was adopted to collect the image dataset. Only a sample with a palm tree located in the center was classified as positive palm tree detection. Spatial coordinates of the detected palm tree classes are obtained and those corresponding to the same palm tree samples are merged. Those spatial coordinates with a Euclidean distance below a certain threshold are grouped into one coordinate. The remaining coordinates represent the actual coordinates of the detected palm trees. The work reported accurate detection of 96% palm trees in the study area.

Faster R-CNN [113] architecture was employed for car detection from low-altitude UAV imagery in [134]. Faster R-CNN comprises a region proposal network (RPN) module and a fast R-CNN detector. The RPN module is a deep convolutional

architecture that generates region proposals of varying scales and aspect ratios. Region proposals may not necessarily contain the target object. These region proposals are further refined by the fast R-CNN detector. The RPN and fast R-CNN detector modules share their convolutional layers and are jointly trained for object detection. For the car detection task, the VGG-16 model [125] was adopted to form the shared convolutional network. The RPN generates k region proposals in the form of $2k$ box classification and $4k$ box regression outputs. The box regression outputs correspond to the coordinates of the k region proposals while the box classification represents the objectness score, *i.e.*, the probability that each proposal contains the target object (car) or not. The faster R-CNN is trained with a multitask loss function comprising of classification and regression components. The car detection imagery was collected with GoPro Hero Black Edition-3 mounted on a DJI Phantom-2 quadcopter. The paper reported car detection accuracy of 94.94% and demonstrated the robustness of the method to scale, orientation, and illumination variations. For a simple exposition, the faster R-CNN architecture is shown in Fig.1.13. In [97], the faster R-CNN

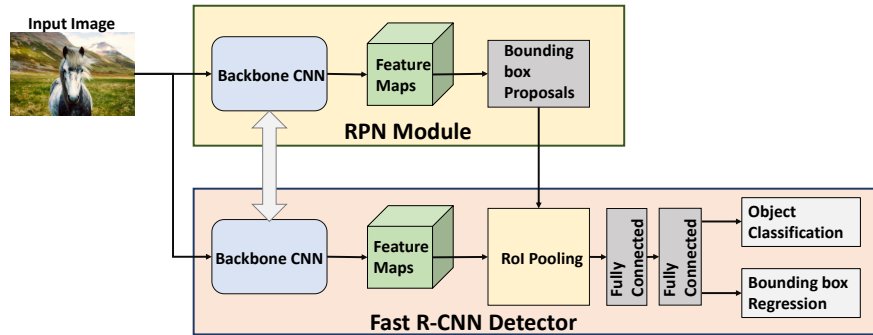


Fig. 1.13 Faster R-CNN architecture

architecture is applied for maize tassel detection from UAV RGB imagery. Here, different CNN architectures were experimented to form the shared layers between the RPN and fast R-CNN detector modules. The paper reported higher accuracy with ResNet [68] in contrast to VGGNet for image resolution of 600×600 and UAV altitude of 15 m.

The faster R-CNN architecture was compared with You Only Look Once (YOLO v3) [112] for car detection from UAV imagery in [46]. YOLOv3 is an advancement over its predecessors YOLOv1 [110] and YOLOv2 [111]. Unlike its predecessors, YOLOv3 can perform multi-label classification of the detected object. Secondly, the bounding box prediction assigns an objectness score of 1 to the predicted box that overlaps the ground truth box more than a predefined threshold. In this way, YOLOv3 assigns one bounding box corresponding to a ground truth object. Additionally, YOLOv3 predicts bounding boxes at 3 different scales. Lastly, it adopts a 53-layered CNN feature extractor named Darknet-53. The study found both YOLOv3 and faster R-CNN performing comparably well in classifying the car ob-

ject from the image. Although YOLOv3 outperformed faster R-CNN in processing time and sensitivity, *i.e.*, the ability to identify all the cars in the image.

In [102], a Partially Shared-Deep Neural Network (PS-DNN) is used for voice identification of people for emergency rescue missions. The microphone array embedded onboard a Parrot Bebop UAV is used for collecting acoustic data. The PS-DNN is posed as a multitask learning framework to achieve two simultaneous tasks - sound source separation and sound source identification. The PS-DNN for multitask learning is a feedforward neural network with partially shared hidden layers between the two sub-networks. Mel filter bank feature vectors obtained by applying windowed Short Time Fourier Transform (STFT) on the acoustic signals are fed as input to the PS-DNN. The network was trained with Adam learning optimizer [82] with a learning rate of 2×10^{-4} . The study demonstrated promising accuracy when a partially annotated dataset was employed.

Three Event Sound Classification (ESC) models - CNN, RNN, and Gaussian Mixture Model (GMM) - were experimented in [78] to detect commercial drones in real noisy environments. The dataset consisted of ordinary real-life noises and sounds from commercial drones such as 3DR Solo, DJI Phantom-3, DJI Phantom-4, and DJI Inspire. The study demonstrated RNN outperforming the CNN and GMM models. The RNN architecture is a bidirectional LSTM with 3 layers and 300 LSTM units. An early-stopping strategy is adopted in the training phase such that if the accuracy and loss do not improve after 3 epochs, the training is stopped. RNN exhibited good generalization over unseen data types with an F-score of 0.6984 and a balanced precision and recall while the CNN resulted in false positives. On the other hand, GMM exhibited better detection performance to CNN but low F-scores deterring practical use.

Drone identification based on acoustic fingerprints using CNN, RNN, and Convolutional Recurrent Neural Network (CRNN) is presented in [40]. CRNN [41] exploits the advantages of both CNN and RNN to extract spatio-temporal features. The three different architectures were utilized to extract unique acoustic signatures of the flying drones. The authors collected the drone acoustic dataset by recording the sound produced by the drone's propellers while flying them in an indoor environment. Two types of UAVs from the Parrot family named Bebop and Mambo were utilized in this study. The neural networks classify the audio input as drone and not drone. The work portrayed the CNN outperforming both RNN and CRNN in terms of accuracy, precision, F1-score, and recall while RNN exhibited lesser training time. However, the performance of RNN was very poor on all counts which could be attributed to the short duration audio clips as opposed to long sequential data. CRNN, however, outperformed RNN and exhibited comparable performance to that of CNN with the added benefit of lesser training time. The authors also extended their work to multi-label classification to identify the audio clips as Bebop, Mambo, and Unknown. In this task again, a similar performance trend was observed as with the binary classification.

1.3.2 UAS Path Planning and Situational Awareness

A CNN-based controller strategy for autonomous indoor UAV navigation is considered in [80]. The limited precision of Global Positioning System (GPS) in the indoor environment and the inability to carry heavy weight sensors render indoor navigation a challenging task. The CNN aims to learn a controller strategy to mimic an expert pilot's navigation decisions. The dataset of seven unique indoor locations was collected with a single forward facing camera onboard a Parrot Bebop Drone. The classifier is trained to return flight commands - Move Left, Move Right, Move Forward, Spin Left, Spin Right, and Stop - by training with manually labeled expert flight commands. The CNN classifier followed the CaffeNet [79] architecture with five convolutional layers and three fully connected layers. The classifier was trained on NVIDIA GTX 970M Graphics Processing Unit (GPU) with NVIDIA cuDNN [53]. The trained classifier is tested on a combination of familiar and unseen test environments with different objects, lighting, and geometry. The classifier reported success rates in the range of 60%-80% for the test locations implying acceptable robustness in flying autonomously through buildings with different objects and geometry.

An interesting approach to UAV navigation is adopted in [62] where it is taught to fly by crashing. Here, the authors create a crash dataset by crashing the UAV under different scenarios 11500 times in addition to non-crash data sampled from the same trajectories. In other words, the drone is allowed to learn not to collide into objects by crashing. The collision data is collected by placing the Parrot AR. Drone 2.0 in a random location which is then allowed to takeoff in a random direction and follow a straight line path until the collision. This way the model is allowed to learn if going straight in a specific direction is good or not. The network architecture adopted the AlexNet [88] pre-trained on ImageNet [57]. The pre-trained weights act as initialization for the network weights rather than randomly initialized weights except for the last fully connected layer. The AlexNet architecture involves five convolutional layers and three fully connected layers. The final layer adopts the binary softmax activation function which classifies the navigational actions for the drone. Given an input image, the network decides whether to go left, right or straight. Experimental demonstrations portrayed the efficacy of this supervised learning approach in avoiding glass walls/doors, corridors, and hallways in contrast to an image depth estimation method.

A regression CNN for indoor navigation is proposed in [87]. Autonomous indoor navigation is enabled by predicting the distance to collision based on the visual input from the monocular camera onboard. The authors adopt a self-supervised approach to collect indoor flight dataset annotated with distance to the nearest obstacle in three different diverging directions. The automated annotation is enabled with the help of three pairs of infrared and ultrasonic sensors mounted on the UAV pointing towards different directions with respect to the camera's field of view. The regression CNN follows a two-stream architecture with the first two layers of the streams similar to that of the AlexNet CNN. The two streams are fused to concatenate the feature maps from the streams followed by processing with a convolutional

layer similar to the third convolutional layer of AlexNet. The two subsequent convolutional layers in the single-stream section also adopt the last two convolutional layers of AlexNet except for the classifier unit in AlexNet which is replaced by a single fully-connected regression layer. The training of the regression CNN was performed with SGD with momentum in 30 epochs with a mini-batch size of 128. The implementation and training were performed in MATLAB on a desktop server with Intel Xeon E5-2630 processor, 64GB of RAM, and a GTX1080 GPU. The UAV is a Parrot AR-Drone 2.0 with a 720p forward-facing camera onboard. During the experiments, a WiFi connection is established between the UAV and a laptop with an Intel Core i7-6700HQ, 16GB of RAM, and a GTX1070 GPU to perform the CNN inference and motion planning. The authors compared the performance of the proposed regression CNN against two previously discussed state-of-the-art schemes [80] and [62]. Regression CNN demonstrated continuous navigation time without collision $4.6\times$ and $1.7\times$ more compared to [80] and [62] respectively.

A MAV-assisted supervised deep learning approach for ground robot path planning to perform search and rescue operation is proposed in [56]. The path planning is executed in three stages. The initial stage involves a human operator flying the MAV in vision-assisted mode to localize a goal location such as a ground robot or a victim. During this initial flight, the camera imagery from the MAV is collected for initial terrain classification. The terrain is mapped to obtain a precise elevation map by monocular 3D reconstruction. The CNN classifier is trained on-the-spot without any *a priori* information. The on-the-spot classifier training involves an operator flying the MAV and labeling a few regions of interest from the live camera imagery. Many training patches are gathered from the few labeled regions by cropping patches that fall on previously labeled areas. The authors of [56] also report a spot training time of 10 - 15 min on a CNN. Post training, the patches are classified and projected on to the terrain map. After the goal location is found, the second stage involves an autonomous vision-guided flight to a series of waypoints. The path exploration follows an exhaustive search over the candidate paths in order to effectively reduce the response time. The authors demonstrated the efficacy of their approach via simulation as well as field trials. The MAV for field trials was custom built with onboard Inertial Measurement Unit (IMU), quadrotor, downward facing camera, onboard Odroid U3 quad-core computer, and PIXHAWK autopilot software. The ground robot for the experiment was a Bluebotics Absolem which is capable of driving over rough terrain. The field trials with canyon and driveway scenarios demonstrated feasible and efficient path exploration over multiple terrain classes, elevation changes, and untraversable terrain.

Another CNN architecture - whereCNN - was proposed in [95] to perform ground to aerial geolocalization. The method aims at mapping a street-view query image to its corresponding location on a city-scale aerial-view image. The CNN architecture for cross-view image matching is inspired by Siamese network [54] and is comprised of two identical CNNs to learn a shared deep representation across pairs of street and aerial view images. A contrastive loss function is used as the overall loss to train the whereCNN such that the matched pairs are penalized by their squared Euclidean distance and the mismatched pairs by the squared Euclidean dis-

tance to a small margin (for the distance that is smaller than the margin). A smaller margin causes the network to be influenced by harder negatives. The dataset is comprised of 78k pairs of Google street view images along with their corresponding aerial view. The whereCNN was trained for 4 days on an NVIDIA Grid K520 GPU. The authors demonstrated that the whereCNN trained without sharing parameters between the siamese network entities generalizes reasonably well on unseen data. The method exhibited cross-view matching accuracy of over 22% for Charleston, San Diego, and San Francisco.

In Table 1.1, we summarize the deep learning techniques that enable autonomous UAV applications.

Table 1.1 Deep learning for UAV autonomy

Proposed solution	Architecture	Application
Giusti et al. [63]	CNN	Outdoor UAV navigation
Li et al. [93]	AlexNet	Palm tree detection and counting
Xu et al. [134]	Faster R-CNN	Car detection from low-altitude UAV imagery
Liu et al. [97]	Faster R-CNN	Maize tassel detection
Benjdira et al. [46]	Faster R-CNN, YOLOv3	Car detection from UAV imagery
Morito et al. [102]	PS-DNN	Emergency rescue mission
Jeon et al. [78]	RNN	Drone identification
S. Al-Emadi et al. [40]	CNN, RNN, CRNN	Drone identification
D. K. Kim and T. Chen [80]	CaffeNet	Indoor UAV navigation
Gandhi et al. [62]	AlexNet	Indoor UAV navigation
A. Kouris and C. Bouganis [87]	CNN	Indoor UAV navigation
Delmerico et al. [56]	CNN	UAV-assisted ground robot navigation
Lin et al. [95]	whereCNN	Ground to aerial geolocation

1.3.3 Open Problems and Challenges

In this section 1.3, we discussed the state of the art deep learning techniques for achieving various UAS tasks. Specifically, we discussed how deep learning can be leveraged to accomplish feature extraction from sensor data, planning, and situa-

tional awareness. However, there exist several open research challenges on the road to achieving complete autonomy of UAS tasks. A few of these are enlisted below:

1. *Lack of realistic datasets*: The realistic gap between simulated and actual deployed scenarios poses a severe challenge to the deployed deep learning solutions. The diverse scenarios that can be confronted by a UAV in a realistic setting in terms of the varied obstacles in the traversed path, occluded or visually artifacted targets in an object detection task, the effects caused by the sensors on board, etc., are hard to model in a virtual setting. In addition, generating such a realistic dataset from actual UAVs followed by annotating them is a laborious task.
2. *Fast deep learning*: Generalizing a supervised deep learning solution to unseen data such as those not represented by the training dataset is an open research challenge. On-the-spot learning implying training of the neural network on-the-fly with limited snapshots of the scenario will prove useful in allowing the model to continue learning new scenarios without forgetting past knowledge. The recently introduced model agnostic meta learning (MAML) [60] opens door to developing such fast learning techniques.
3. *Resource-heavy deep learning techniques*: The computational complexity of deep learning architectures is another significant hurdle that poses severe constraints on the latency, weight, flight time, power consumption, and cost. Denser architectures require powerful computational platforms such as GPUs that are often above the prebuilt onboard computational capacity of the UAVs requiring auxiliary computational units. Such additional computational platforms increase the cost, weight, flight time, and power consumption of the UAVs.
4. *Vulnerability to cyberattacks*: Vulnerability of the deployed deep learning techniques to various security attacks is a cause of serious concern. Spoofing attacks, signal jamming, identity forging, among others can disrupt the intended UAV operation leading to asset loss and damage. Integrating adversarial learning techniques to the application-specific deep learning approaches can be one way to tackle such security threats.

1.4 Reinforcement Learning for UAS Autonomy

Reinforcement learning provides a learning framework allowing agents to act optimally via sequential interactions with its environment. In comparison to supervised or unsupervised learning, reinforcement learning allows the agent to leverage its own experiences derived from environmental interactions. Additionally, reinforcement learning provides a means to specify goals for the agent by means of a reward and penalty scheme. These characteristics of reinforcement learning have led to many research efforts on its application to autonomous UAS applications. Reinforcement learning has been primarily applied to lower-level control system tasks that regulate the UAV's velocity, attitude, and navigation as well as other higher-level tasks.

1.4.1 UAS Control System

Stable control of a UAS is a complex task due to nonlinear flight dynamics. Traditional control approaches such as Proportional-Integral-Derivative (PID) controllers have been successfully used for UAS for attitude and velocity control in stable environments. However, the performance of these controllers can deteriorate in dynamic or harsh environments. The main disadvantages of PID control being a constant parameter feedback controller are the control efforts are reactive and the controller does not have a priori knowledge of or the ability to learn about the environment. Techniques from adaptive and robust control can provide insights on designing controllers that can adapt to dynamic environments and operate effectively in the presence of uncertainties. However, a shortcoming of these traditional control techniques is that they typically require a mathematical model of the environmental dynamics and do not explicitly learn from past experiences. Reinforcement learning algorithms present a potential solution to the problem of UAS control due to their ability to adapt to unknown environments.

There have been many research efforts focusing on the application of reinforcement learning to control systems on a UAS [133, 47, 58, 135, 72, 90, 85, 49]. Much of the research has been focused on quadrotor UAVs; however, some of the early works involved autonomous helicopters. Many of the reinforcement learning based control systems discussed in this section are for attitude control of the UAV but some of the works consider trajectory tracking and maneuvering as well. Additionally, several algorithmic approaches have been studied including both online and offline methods operating in conjunction with traditional control algorithms as well as DRL based approaches.

Early works of applying reinforcement learning to UAV control problems focused on autonomous helicopters [45, 81, 104, 37]. In these works, data was collected from a human pilot flying a remote control helicopter and the dynamics were learned offline. From the learned dynamics, reinforcement learning algorithms were used to design controllers for various maneuvers including hovering, trajectory tracking, and several advanced maneuvers including inverted hovering, flips, rolls, tunnels, and others from the Academy of Model Aeronautics (AMA) remote control helicopter competition.

The first work that used reinforcement learning for quadrotor UAV control did so for altitude control [133]. A model-based reinforcement learning algorithm that rewards accurate tracking and good damping performance was utilized to find an optimal control policy. To benchmark with a traditional approach, an integral sliding mode controller was also implemented. Tests conducted on Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC) quadrotors showed both the reinforcement learning and integral sliding mode controllers to have comparable performance, both significantly exceeding that of traditional linear control strategies.

In [47], Fitted Value Iteration (FVI) is used to design a velocity control system for a quadrotor UAV. The reinforcement learning FVI controller was compared to a cascaded velocity and attitude controller designed using nonlinear control techniques.

The performance of each controller was compared using numerical simulations in MATLAB/SIMULINK. While both controllers produced satisfactory results, the reinforcement learning controller was outperformed in terms of settling time but had a lower percent overshoot. The authors stated that a non-parametric approach to value function estimation, such as the use of a wavelet network, may have resulted in better performance for the reinforcement learning controller. The authors emphasized that an advantage of the reinforcement learning controller is that it does not require any prior mathematical knowledge of quadrotor dynamics to yield satisfactory behavior.

In [58], a Learning Automata reinforcement learning algorithm called Finite Action-set Learning Automata (FALA) was used to learn the optimal parameters of nonlinear controllers for trajectory tracking and attitude control. Traditional approaches such as PID, sliding mode, and backstepping controllers were used to benchmark against FALA. The performance of the controllers was analyzed in simulation under varying non-linear disturbances including wind and ground effects. The reinforcement learning tuned controllers outperformed the mathematically tuned controllers in terms of tracking errors.

In [135], an off-policy method, Model Predictive Control (MPC), is used for guided policy search for a deep neural network policy for UAV obstacle avoidance. During training, MPC is used to generate control actions for the UAV using knowledge of the full state, this is used along with the state observations to train the policy network in a supervised learning setting. During testing, only the state observations are available to the policy neural network. Simulations were conducted that demonstrated that the proposed approach was able to successfully generalize to new environments.

In [72], a neural network based policy trained using reinforcement learning is used for trajectory tracking and recovery maneuvers. The authors proposed a new reinforcement learning method that uses deterministic policy optimization using natural gradient descent. Experiments were conducted in both simulation and on a real quadrotor UAV, the Ascending Technologies Hummingbird, that demonstrated the effectiveness of the proposed approach. In simulations, the proposed method outperformed the popular algorithms TRPO and DDPG. The trajectory tracking test resulted in a small but acceptable steady-state error. Additionally, a recovery test where the quadrotor was manually thrown upside down demonstrated autonomous UAV stabilization. A benefit of the proposed algorithm is low computation time; average time of $6 \mu s$ was reported.

In [90], deep Model Based Reinforcement Learning (MBRL) is used for low-level control of a quadrotor UAV. Deep MBRL is used to learn a forward dynamics model of the quadrotor and then MPC is used as a framework for control. The algorithms were evaluated using a Crazyflie nano quadrotor. Stable hovering for 6 seconds using 3 minutes of training data was achieved emphasizing the ability to generate a functional controller with limited data and without assuming any apriori dynamics model.

In [85], multiple neural network based reinforcement learning algorithms are evaluated for attitude control of UAVs. The algorithms that were evaluated include

DDPG, TRPO, and PPO. The reinforcement learning algorithms were compared against PID control systems for attitude control of UAVs in a simulation environment. The authors also developed an open-source training environment utilizing OpenAI and was evaluated using the Gazebo simulator. The simulations indicated that the agents trained with PPO outperformed a tuned PID controller in terms of the rise time, overshoot, and average tracking error.

In [49], PPO is applied for attitude control of fixed-wing UAVs. The PPO method was chosen largely due to the success reported in [85]. The PPO controller was trained in a simulation environment to control the attitude (pitch, roll) and airspeed of the UAV to the specified setpoints. The results showed that the DRL controller was able to generalize well to environments with turbulence. The advantages of the DRL controller were emphasized in the high turbulence scenarios by it outperforming the PID controller in multiple performance metrics including success percentage, rise time, settling time, and percent overshoot.

A DRL robust control algorithm for quadrotor UAVs is presented in [132]. The algorithm uses DPG which is an actor-critic method. Furthermore, similar to classical control design, DPG is augmented with an integral compensator to eliminate steady state errors. Additionally, a two phase learning protocol consisting of an offline and online learning phase is defined for training the model. The offline training is completed using a simplified quadrotor model but the robust generalization capabilities are validated in simulation by changing model parameters and adding disturbances. The capability of the model to learn an improved policy online is demonstrated with faster response time and less overshoot compared to original policy learned offline.

1.4.2 Navigation and Higher Level Tasks

In this section, the use of reinforcement learning for higher level planning tasks such as navigation, obstacle avoidance, and landing maneuvers is studied.

In [74], a model-based reinforcement learning algorithm is used as a high level control method for autonomous navigation of quadrotor UAVs in an unknown environment. A reinforcement learning algorithm called TEXPLORE [71] is utilized to perform a targeted exploration of states that are both uncertain in the model and promising for the final policy. This is in contrast to an algorithm such as Q-learning that attempts to exhaustively explore the state space. TEXPLORE uses decision trees and random forests to learn the environmental model. In particular, the decision trees are used to predict the relative state transitions and transition effects. A random forest is used to learn several models of the environment as a single decision tree may learn an inaccurate model. The final model is averaged over the decision trees in the random forest. TEXPLORE then performs its targeted exploration using an algorithm called Upper Confidence bounds applied to Trees (UCT). The authors implement and compare the TEXPLORE algorithm to Q-Learning for a navigation task. The navigation task involves the UAV traveling from a start to an end state

under battery constraints i.e., the UAV requires a recharge during the mission in order to make it to the goal. The navigation task is performed in a simulated grid environment implemented using ROS and Gazebo. It is shown that the TEXPLORE algorithm learns effective navigation policies and outperforms the Q-Learning algorithm considerably.

In [106], a PID and Q-Learning algorithm for navigation of a UAV in an unknown environment is presented. The problem is modeled as a finite MDP. The environment is modeled as a finite set of spheres with the centers forming a grid, the state of the UAV is its approximate position i.e. one of the points on the grid, and the actions available to the agent are head North, South, East, or West. In this work, a constant flight altitude is assumed and thus the state space is two dimensional. The objective of the agent is to navigate to a goal position following the shortest path in an unknown environment. A PID and Q-Learning algorithm are used in conjunction to navigate the UAV to the goal position in the unknown environment. The Q-Learning algorithm and ϵ -greedy policy are used by the agent to select the next action given the current state. The action is then translated to a desired position and is inputted to the PID controller which outputs control commands to the UAV to complete the desired action. The proposed algorithm was implemented and tested in both simulation and on a Parrot AR Drone 2.0. In both simulation and experimentation, the UAV was able to learn the shortest path to the goal after 38 episodes.

In [107], the authors of [106] utilize an approximated Q-Learning algorithm that employs function approximation in conjunction with the previously described PID and Q-Learning control algorithm for UAV navigation tasks. Function approximation is used to handle the large state space and to provide faster convergence time. Fixed sparse representation is used to represent the Q table as a parameter vector. Compared to the work in [106], the state representation consists of the relative distance of the UAV to the goal and relative distances to obstacles in four directions obtained using on-board radar. Both simulation and real tests demonstrated faster convergence and UAV navigation to the goal position.

In [131], the authors introduce a DRL algorithm, a variant of RDPG called Fast-RDPG, for autonomous UAV navigation in large complex environments. The Fast-RDPG differs from RDPG as it uses non-sparse rewards allowing for the agent to learn online and speed up the convergence rate. The reward function design is discussed which includes transition (i.e., progress towards the goal), obstacle proximity penalty, free space, and time step penalty. The Fast-RDPG algorithm outperforms RDPG and DDPG in terms of rate of success, crash, and stray metrics. Generalization of the Fast-RDPG algorithm to environments of different sizes, different target altitudes, and 3D navigation is discussed as well.

In [126], a Deep Recurrent Q-Network with temporal attention is proposed as a UAV controller for obstacle avoidance tasks. The model uses a conditional generative adversarial network to predict a depth map from monocular RGB images. The predicted depth map is then used to select the optimal control action. The temporal attention mechanism is used to weight the importance of a sequence of observations over time which is important for obstacle avoidance tasks. The performance of the proposed approach was compared to Deep Q-Network, D3QN, and Deep Recurrent

Q-Network without temporal attention algorithms and showed superior performance in simulations.

In [114], a DRL algorithm called Deep DPG is used to enable an advanced autonomous UAV maneuvering and landing on a moving platform. The authors integrate the Deep DPG algorithm into their reinforcement learning simulation framework implemented using Gazebo and Robot Operating System (ROS). The training phase of the proposed approach was conducted in simulation and the testing phases were conducted in both simulation and real flight. The experiments demonstrated the feasibility of the proposed algorithm in completing the autonomous landing task. Additionally, this work showed that agents trained in the simulation are capable of performing effectively in real flights.

In [108], a DRL based approach to perform autonomous landing maneuver is presented. The approach relies on a single downward facing camera as the sole sensor. The landing maneuver is considered as a three phase problem: landmark detection, descent maneuver, and touchdown. A hierarchy of two independent DQNs is proposed as a solution for the landmark detection and descent maneuver problems. The touchdown maneuver is not considered in the research; however, the authors indicated that it may be solved using a closed loop PID controller. A DQN is employed for the landmark detection component and a double DQN is used for the descent. Additionally, the authors propose a new form of prioritized experience replay called *partitioned buffer replay* to handle sparse rewards. Various simulations were conducted that indicated that the proposed DRL approach was capable of performing the landing maneuver and could effectively generalize to new scenarios.

In Table 1.2, we summarize the reinforcement learning techniques that enable autonomous UAV applications.

1.4.3 Open Problems and Challenges

There are still several open problems and challenges associated with reinforcement learning based autonomous UAV solutions. Many problems and challenges are associated with the transition from simulation to hardware. This is evidenced by limited results on the performance of reinforcement learning solutions performing high complexity planning tasks in real life tests. A challenge associated with the transition is managing the reality gap between simulation and real life testing. Additionally, as deep reinforcement learning solutions are utilized for autonomy, the integration onto an embedded UAV platform can become challenging due to the computational requirements of the algorithms and the Size, Weight, and Power (SWaP) constraints of the UAV.

Other challenge areas include developing algorithmic solutions that enable higher degrees of autonomy. For example, more complex tasks and missions may require the UAV to cooperate with other autonomous systems and/or humans via Natural User Interfaces (NUIs). Also, the majority of the published works consider scenarios with a static mission objective in dynamic environments; however, in general,

Table 1.2 Reinforcement for UAS autonomy

Proposed Solution	Reinforcement Learning Technique	Application
J. A. Bagnell and J. G. Schneider [45]	Model-based, PEGASUS	Helicopter control
Kim et al. [81]	Model-based, PEGASUS	Helicopter hovering and maneuvers
Ng. et. al.[104]	Model-based, PEGASUS	Helicopter inverted hovering
Abbeel et. al. [37]	Differential Dynamic Programming	Helicopter aerobatic maneuvers
S. L. Waslander and G. Hoffmann [133]	Model-based; Locally Weighted Linear Regression (LWLR), Policy Iteration	Quadrotor altitude control
Bou-Ammar et. al. [47]	Fitted Value Iteration	Quadrotor velocity control
S. R. B. dos Santos et. al. [58]	Finite Action-set Learning Automata	Quadrotor trajectory tracking and attitude control
Zhang et. al. [135]	MPC Guided Policy Search	Quadrotor obstacle avoidance
Hwangbo et. al. [72]	Neural network policy	Waypoint tracking and recovery tests
Lambert et. al. [90]	Deep model-based	Hovering
Koch et. al. [85]	DDPG, TRPO, PPO	Attitude control
Bøhn et. al. [49]	PPO	Attitude control
Y. Wang et. al. [132]	DPG	UAV control
Imanberdiyev et. al. [74]	Model-based,TEXPLORE	UAV navigation
Pham et. al. [106]	Q-Learning	UAV navigation
Pham et. al. [107]	Q-Learning with function approximation	UAV navigation
C. Wang et. al. [131]	Fast-RDPG	UAV navigation
Singla et. al. [126]	Deep recurrent Q network with temporal attention	Obstacle avoidance
A. Rodriguez-Ramos et. al. [114]	DDPG	Landing on a moving platform
Polvara et. al. [108]	DQN	Autonomous landing

the autonomous agent will need to be able to operate in scenarios where both mission objectives and the environment are dynamic. It is also possible that the mission will consist of multiple objectives that need to be completed simultaneously.

1.5 Simulation Platforms for UAS

The ability to accurately simulate UAS in realistic operational environments is an invaluable capability. This is largely due to the fact that real hardware-based testing of UAS is both a time consuming and expensive process. The potential for injuries and damages or losses are the main challenges associated with hardware-based testing. Additional challenges and constraints include limited battery life and the laws and regulations of outdoor flight. These challenges are exacerbated in the context of deep learning and reinforcement learning based autonomy solutions as they require large amounts of training data and experiences in order to learn effective behaviors and are also often unstable during their training phases. Additionally, it can also be challenging and/or costly to collect ample training data for machine learning based autonomous UAS algorithms. Physically and visually realistic UAS simulations are potential solutions to several of these challenges. For example, a realistic visual simulation of an operational environment could be used to create a dataset for a deep learning algorithm. Furthermore, simulation provides a means to test UAS in scenarios that can be hard to create in real life e.g. failure modes, harsh environmental conditions, etc. Simulation also provides a means for establishing easily repeatable environments for algorithm comparisons and software regression testing.

1.5.1 Simulation Suites

This section now presents a survey of popular simulation software platforms for UAS. Previous surveys conducted in [59, 70, 99] introduced the majority of available UAS simulation platforms for various applications. The discussion in this section focuses primarily on open-source simulators that appear useful for research and development of autonomous UAS applications.

Gazebo [13, 86] is an open-source robotics simulator capable of simulating multiple robots in both indoor and outdoor environments. This is enabled by its integration with high-performance physics engines, e.g., Open Dynamics Engine (ODE), Bullet, Simbody, and Dynamic Animation and Robotics Toolkit (DART) as well as its ability to model various sensors, noise, and environmental effects. The Gazebo architecture is modular by allowing for worlds and objects to be defined using Simulation Description Format (SDF) files while enabling sensor and environmental effect modules to be added as plugins. Object-Oriented Graphics Rendering Engine (OGRE) [22] is utilized by Gazebo for high fidelity visual rendering of the environment that captures different textures and lighting. Gazebo is also one of the default simulators integrated with the popular robotics middleware package ROS. By itself, Gazebo does not provide the capability to simulate UAVs; however, there have been multiple works that define the necessary model, sensor, and controller plugins to facilitate UAV simulation and is discussed herein. An example of UAV simulation using Gazebo is shown in Figure 1.14.

In [100, 15] simulation of quadrotor UAVs using Gazebo and ROS is implemented as an open-source package called `hector_quadrotor`. The `hector_quadrotor` package provides the geometry, dynamics, and sensor models for quadrotor UAVs. Sensor models for IMU, barometer, ultrasonic sensor, magnetic field, and GPS in addition to the default sensor models provided by Gazebo such as LIDAR and cameras. Extended Kalman Filters (EKFs) and cascaded PID controllers are implemented and utilized for state estimation and control respectively. A tutorial example of integrating a LIDAR based Simultaneous Localization and Mapping (SLAM) algorithm with the simulated UAVs is included in the package's documentation.

RotorS is another open-source MAV simulator using Gazebo and ROS [61, 28]. Models of various multirotor UAVs including the AscTec Hummingbird, AscTec Pelican, and AscTec Firefly are included with the simulator. Default simulator sensors include IMU, a generic odometry sensor, and visual inertial sensor. Similar to `hector_quadrotor`, RotorS provides a baseline UAV simulation using Gazebo by defining the required UAV, sensor, and controller configuration files and plugins. The RotorS package provides a well documented and functional UAV simulator that a researcher can use for rapid prototyping of new autonomous UAV control algorithms.

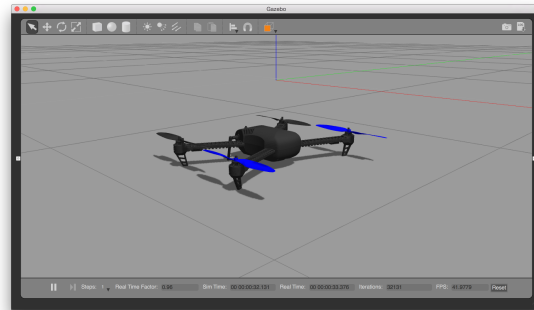


Fig. 1.14 UAV simulation in Gazebo [14]

In [85], a framework called GymFC for tuning UAV flight control systems was introduced. The framework integrates the popular reinforcement learning toolkit OpenAI Gym [48] and the Gazebo simulator to facilitate research and development of attitude flight control systems using DRL. GymFC defines three layers to provide seamless integration of reinforcement learning based UAV control algorithms: Digital Twin Layer, Communication Layer, and Environment Interface Layer. The Digital Twin Layer consists of the simulated UAV and environment as well as interfaces to the Communication Layer. The Communication Layer is the interface between the Digital Twin and Environment Interface Layer that implements lower level functionality to enable control of the UAV and the simulation. The Environment Interface Layer implements the environmental interface defined by the OpenAI Gym API that the reinforcement learning agent interacts with. In the original

work [85], the proposed DRL based attitude controllers were only evaluated in simulation. The open-source Neuroflight framework [84] has since been introduced for deploying neural network based low-level flight control firmware on real UAVs. Neuroflight utilizes GymFC for initial training and testing of controllers in a simulation environment and then deploys the trained models to the UAV platform. Initial tests of Neuroflight have demonstrated stable flight and maneuver execution while the neural network based controller runs on an embedded processor onboard the UAV.

The Aerostack software framework [117, 3, 116] defines an architectural design to enable advanced UAV autonomy. Additionally, Aerostack has been used for autonomous UAV research and development in both simulations (utilizing the RotorS simulator [61]) and in hardware such as the Parrot AR Drone.

Microsoft AirSim [123, 4] is an open-source simulator for both aerial and ground vehicles. AirSim provides realistic visual rendering of simulated environments using the Unreal Engine, as shown in Figure 1.15. AirSim was designed as a simulation platform to facilitate research and development of Artificial Intelligence (AI) enabled autonomous ground and aerial vehicles which motivates its use when developing deep learning and reinforcement learning UAS solutions. The software is cross platform and can be used on Linux, Windows, and Macintosh operating systems. The AirSim software comes with extensive documentation, tutorials, and Application Programming Interfaces (APIs) for interfacing with vehicles, sensors, and environment for programmatic control and data collection for model training. Recently, AirSim was used as a platform to host a simulation-based drone racing competition called Game of Drones [98].



Fig. 1.15 UAV simulation in AirSim [5]

A final consideration is that the popular flight control stacks - PX4 and ArduPilot (discussed in detail in section 1.6.2) - can both integrate with Gazebo and AirSim for software-in-the-loop and hardware-in-the-loop simulations. The Gazebo interfaces maintained by PX4 are derived from the RotorS project.

1.5.2 Open Problems and Challenges

Even with the advances made in the realm of UAS simulations, there are still multiple problems and challenges associated with it. The first problem is typical to any open-source platforms used in different domains - there is no official or industry accepted standard platform. For example, the two most popular open-source flight control stacks, ArduPilot and PX4, both support multiple simulators but there is not a specified official/default simulator common to them. At this time, it appears that both Gazebo or AirSim have the potential for use in autonomous UAS research and development. A challenge associated with the Gazebo simulator is that although it is widely used in UAS simulation, it technically does not provide native UAS simulation support. Works such as [100, 61] implement the required plugins, configuration, and baseline controllers to enable UAV simulation using Gazebo. Additionally, as common with open-source software, there is often limited software maintenance, development support, and documentation of the open-source simulators.

An additional challenge associated with UAS simulation is that there can be steep learning curves associated with advanced usage and software development. It appears to be straightforward to install and run examples provided by the simulator; however, it may take time to familiarize with simulator configurations, development workflow, and software APIs. For example, a developer may be required to add support for a new UAV platform, sensor type, or environment tailored for the research application. This problem could be mitigated to an extent as the use of these platforms become widespread and if there is a uniform standard to add new features that can be made available to the community.

An open problem is assessing the reality gap between simulation and real life deployment. This problem will be further studied as research and development of algorithms for autonomous UAS continues. Other open problems are associated with the seemingly limited consideration of UAV swarm operation, human interaction via NUIs or ground control stations, and communication systems utilized by the UAS.

1.6 UAV Hardware for Rapid Prototyping

Rapid UAS hardware-based prototyping is an essential step in deploying and validating machine learning solutions. Certain factors such as the unique requirements of the deep learning solution and the cost of commercial-off-the-shelf (COTS) UAS in commercial market are the driving factors in choosing the custom prototyping route. The requirements of deep learning solutions could be unique to the problem under consideration and consequently the needs would vary. For instance, an object detection task might require a stable flight platform with good quality image sensor. However, a target tracking or acoustic-based search and rescue mission might require maneuverable platform with image sensor and acoustic sensors onboard respectively. UAS prototyping for testing deep learning solutions involve several steps such as choosing the appropriate hardware platform, sensors, com-

putational resources, memory unit, flight controller software, among others which depend on the size, weight, and onboard carrying capacity of the UAS platform. This section will serve as a comprehensive guide in choosing the appropriate UAS platform, flight stack software, computational resources as well as the various challenges incurred in UAS prototyping.

1.6.1 Classification Choice

UAVs are classified based on their wings, size, landing, etc., as seen in the beginning of the chapter (section 1.1.2). In this section, however, we will focus on the Fixed-wing and Rotary-wing UAVs. The various UAV classifications will guide the reader in understanding the nuances of the platforms in terms of its hovering, maneuvering, payload capabilities, among others allowing application-specific selection.

Fixed-wing UAV has rigid wings with airfoil allowing it to produce the desired lift and aerodynamics by deflecting the oncoming air. Although they cannot hover at a place and maintain low speed, they support long endurance flights. Further, they require an obstruction-free runway to take-off and land. However, in comparison to rotary-wing, they carry heavier payloads and are energy-efficient owing to their gliding characteristic. The MQ-9 Reaper is an example of a fixed-wing UAV as in Fig. 1.16.

Rotary-wing UAV possesses two or more rotary blades positioned around a fixed mast to achieve the desired aerodynamic thrust. Rotary-wing platforms are capable of hovering tasks, low-altitude flights, and perform Vertical Takeoff and Landing (VTOL). In contrast to fixed-wing, they present flexible maneuverability advantages owing to the rotary blades. Rotary-wing UAVs are further classified into single-rotor, multi-rotor, and fixed-wing hybrid [52].



Fig. 1.16 Fixed Wing UAV [32]



Fig. 1.17 Single-Rotor UAV [124]

Single-rotor UAVs rely on a single front rotor to stay airborne. Although, they possess a tail rotor to control the heading as in Fig. 1.17, it does not count towards the rotor count. The required airflow to move forward is generated by the rotor blades. They are also capable of VTOL and hovering tasks. Since they rely on a singular rotor to stay elevated, the blades are usually longer. In contrast to multi-rotor UAVs, they can carry heavier payloads and are energy-efficient owing to

lesser power requirements for a single rotor. The energy-efficient operation enables longer flight times when compared to multi-rotor platforms. Therefore, single-rotor platforms might present themselves as beneficial for aerial surveying applications which require carrying heavier payloads and extended flight times. Helicopters are an example of single-rotor UAVs.



Fig. 1.18 Multi-Rotor UAV



Fig. 1.19 Fixed Wing Hybrid UAV [31]

Multi-rotor UAVs, on the other hand, uses multiple rotor blades to achieve the desired aerodynamic thrust for lifting and propelling as in Fig. 1.18. Most common examples of this category are tricopter, quadcopter (quadrotor), hexacopter, and octocopter. Multi-rotor platforms can perform complex maneuvering and hovering tasks but have limited payload capability and flight endurance. They also provide a stable platform for aerial inspection, photography, and precision agriculture applications.

Fixed-wing hybrid UAV platforms combine the aerodynamic benefits of fixed-wing and rotary-wing UAV classes (Fig.1.19). This coupling adds the VTOL, hovering, increased flight speed, and long endurance capabilities. Owing to the fairly recent arrival of the hybrid class, there are still very few developmental resources available for this class. The discussion in this section will enable the developer in choosing the appropriate UAV platform tailored to meet the requirements pertinent to their unique machine learning solution.

Build or Buy

Here, we will ponder upon the pros and cons of buying versus building a UAV. Commercial UAVs available in the market would serve as an easier and cost-friendly option to rapidly test deep learning solutions. However, specific mission requirements might urge towards building a custom model.

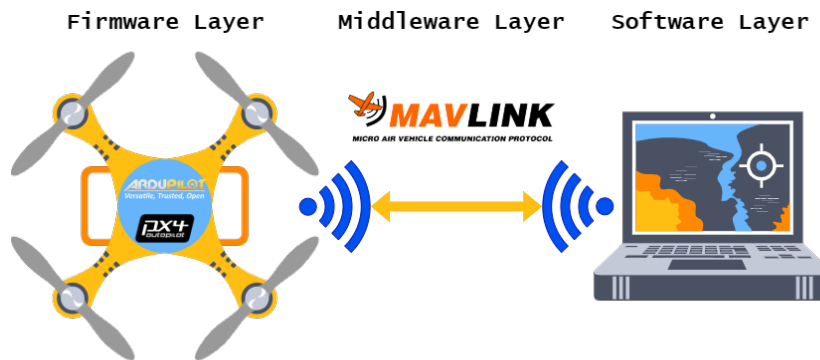
Commercial UAVs are often preprogrammed and tested for stability. Most of them come in a ready-to-fly state requiring minimal setup out of the box. The prebuilt UAVs offer limited customization and could be difficult to repair and/or replace components. An essential requirement for deep learning solutions is the computational power, however, prebuilt UAV platforms have limited onboard computational

resources requiring external processors. A costlier option could be purpose-built commercial UAVs with custom attachments to fit the mission requirements.

UAV prototyping, on the contrary, offers several benefits. Often developers can add custom sensors, batteries, and computational units to a flight-ready UAV platform for rapid deployment and testing. The lift and payload capacity of the UAV judges its flight endurance and stability. Achieving flight stability is guided by several factors such as the right component balance and the ground controller's pilot skills. Building a flight-ready UAV would entail requiring immense electrical and mechanical skills which could be envisioned as a pro as well as a con. The prototyping procedure could be time-consuming while garnering the electro-mechanical skills would be knowledgeable. Another major requirement while building custom prototypes would be the flight controller software needed to control and navigate the UAVs. To conclude, we have listed a few commercial drones and their specifications in Table 1.3. The next subsection sheds light on the flight stack software.

1.6.2 Flight Stack

Flight stack is the flight controller software that comprises of a set of positional, navigational guidance and control algorithms, interfacing, and communication links that directs the UAV's flight path and maneuverability. A flight stack is typically comprised of firmware, middleware, and interface layers as in Fig. 1.20 [50] whereby the middleware supports the communication link to enable command and control (C2) and telemetry data message passing. The software layer performs the interfacing of the firmware via the communication link protocol. Software layer refers to the Ground Control Station (GCS) software that performs UAV configuration and monitoring.



This image was made using resources from Freepik.com

Fig. 1.20 Flight Stack

Table 1.3 Commercially available drones

UAV platform	Specifications	Onboard/ External DL Processing	SDK	Estimated Cost
Ryze Tello EDU [29]	87 g Weight, 13min Flight, WiFi 802.11n, Range Finder, Barometer LED, Camera	External via SDK	Tello-Python	\$129.00
DJI Inspire 2 [7]	3.44 kg Weight, 4.25 kg payload, 27 min flight time, 2.4000 GHz-2.4835GHz, 5.725 GHz-5.850GHz, GPS, GLONASS, GALILEO, Camera, Vision systems for obstacle avoidance	External via SDK	Mobile SDK	\$3,299.00
DJI Matrice 100 [8]	2.355 kg weight, 3.6 kg payload, 13 - 40 min flight time, 5.725-5.825 GHz, 922.7MHz-927.7 MHz, 2.400-2.483 GHz (Lightbridge)	Onboard via Manifold 2-C or Manifold 2-G	Onboard SDK, Mobile SDK	N/A
DJI Matrice 200 Series V2 [18]	4.91 kg weight, 1.23 kg payload, 33 min flight time, 2.4000-2.4835 GHz, 5.725-5.850 GHz, Different Payload configurations	Onboard via Manifold 2-C or Manifold 2-G	Onboard SDK Payload SDK Mobile SDK	Request Quote
DJI Matrice 300 RTK [19]	6.3 kg weight, 2.7 kg payload, 55 min flight time, 2.4000-2.4835 GHz, 5.725-5.850 GHz, Camera Gimbal, infrared ToF Sensing System, FPV Camera, GPS	Onboard via Manifold 2-C or Manifold 2-G	Onboard SDK Payload SDK Mobile SDK	Request Quote
DJI Matrice 600 Pro [9]	9.5 kg weight, 15.5 kg payload, 16 - 38 min flight time, 920.6 MHz-928 MHz, 5.725 GHz-5.825 GHz, 2.400 GHz-2.483 GHz, Camera Gimbal, Collision avoidance system, GPS, GLONASS	Onboard via Manifold 2-C or Manifold 2-G	Onboard SDK, Mobile SDK	\$5,699.00
DJI Mavic 2 Enterprise [21]	905 g weight, 1100 g payload, 29 min flight time, 2.400-2.4835 GHz, 5.725-5.850 GHz, GPS, GLONASS, Visual Camera, Omnidirectional Obstacle Sensing, Speaker, Beacon, Spotlight	External via SDK	Mobile SDK, Windows SDK	Request Quote
DJI Mavic 2 Enterprise Dual [21]	899 g weight, 1100 g payload, 29 min flight time, 2.400-2.4835 GHz, 5.725-5.850 GHz, GPS, GLONASS, Thermal Camera, Visual Camera, Camera, Speaker, Omnidirectional Obstacle Sensing, Beacon, Spotlight	External via SDK	Mobile SDK, Windows SDK	Request Quote
DJI Mavic 2 Pro [20]	905 g weight, 31 min flight time, 2.400-2.4835 GHz, 5.725-5.850 GHz, GPS, GLONASS, Pro Camera, Omnidirectional Obstacle Sensing	External via SDK	Mobile SDK, Windows SDK	\$1,599.00
DJI Mavic 2 Zoom [20]	905 g weight, 31 min flight time, 2.400-2.4835 GHz, 5.725-5.850 GHz, GPS, GLONASS, Zoom Camera, Omnidirectional Obstacle Sensing	External via SDK	Mobile SDK, Windows SDK	\$1,349.00
DJI P4 Multispectral [25]	1487 g weight, 27 min flight time, 2.4000 GHz-2.4835 GHz, 5.725 GHz-5.850 GHz, GPS, GLONASS, GALILEO, RGB Camera, 5 monochrome sensors	External via SDK	Mobile SDK	Request Quote
DJI Phantom 4 Pro V2.0 [10]	1375 g weight, 30 min flight time, 2.4000 GHz-2.4835 GHz, 5.725 GHz-5.850 GHz, GPS, GLONASS, GALILEO, RGB Camera, infrared sensors	External via SDK	Mobile SDK	\$1,599.00
DJI Phantom 4 RTK [27]	1391 g weight, 30 min flight time, 2.4000 GHz-2.4835 GHz, 5.725 GHz-5.850 GHz, GPS, GLONASS, GALILEO, RGB Camera, infrared sensors	External via SDK	Mobile SDK	Request Quote
Parrot ANAFI ANAFI Thermal [34]	315 g weight, 26 min flight time, Wi-Fi 802.11a/b/g/n, GPS, GLONASS, Barometer, magnetometer, vertical camera, ultra sonar, 6 axis, IMU, 3 axis accelerometer, 3 axis gyroscope, thermal imaging camera, 4k camera	External via SDK	Parrot Ground SDK	\$1,900.00
Parrot ANAFI USA [35]	500 g weight, 32 min flight time, Wi-Fi 802.11a/b/g/n, GPS, GLONASS, GALILEO, Barometer, magnetometer, vertical camera, ultra sonar, 6 axis, IMU, 4k camera 3 axis accelerometer, 3 axis gyroscope, 32x zoom camera	External via SDK	Parrot Ground SDK	Coming soon
Parrot ANAFI Work [36]	321 g weight, 25 min flight time, Wi-Fi 802.11a/b/g/n, GPS, GLONASS, Barometer, magnetometer, vertical camera, ultrasonar, 6 axis, IMU, 3 axis accelerometer, 3 axis gyroscope, thermal imaging camera, 4k camera	External via SDK	Parrot Ground SDK	\$999.00

There are many open-source flight controller software available today namely; ArduPilot, PX4, Paparazzi, among others. Flight controller software enables autonomous operation capability to specific UAV platforms (airframes). This comprises fault detection and handling, C2 link protocol, battery monitoring, obstacle avoidance, landing, return home features, data logging, among others. The fault detection and handling support features such as landing when missing C2 link, return to home when missing C2 link, automatic parachute release, battery voltage warning, geofence, land/return to home when battery low, safety check for sensor error, etc. Some of the C2 link protocols are MavLink, UAVTalk, XBUS, XBee, FrSky, HoTT, Pulse Position Modulation (PPM), and Lightweight TeleMetry (LTM).

ArduPilot is an open-source flight controller software released under GNU General Public License (GPL) which supports a wide range of vehicles including fixed-wing UAV, multi-rotor UAV, single-rotor UAV, boats, and submarines [6]. It can be run on a Linux-based operating system (OS) allowing support on single-board computers to full PC systems. ArduPilot has a desktop GCS software for mission planning and calibration for Linux, Windows, and Macintosh OS. It also supports MAVLink, FrSky, and LTM communication protocols. ArduPilot additionally supports the usage of multiple radio control receivers for redundancy, failover, and/or handoffs.

PX4 flight controller [23] from DroneCode collaborative project [11] is released under Berkeley Software Distribution (BSD) license and supports both fixed-wing and multi-rotor airframes. PX4 enables operation with QGroundControl GCS software from where the UAV can be configured as well as monitored. Both ArduPilot and PX4 supports satellite waypoint navigation and satellite position hold. ArduPilot and PX4 additionally support stereovision navigation function and follow me autonomous navigation features respectively.

Paparazzi flight controller supports fixed-wing, flapping-wing, hybrid, and multi-rotor airframes and is released for public use under GNU GPL [26]. The GCS software of Paparazzi enables UAV configuration, monitoring, and custom flight plan configuration for navigational control and guidance. The supported C2 link protocols are MavLink, XBee, SBus, and PPM. Paparazzi supports all autonomous navigation features offered by ArduPilot and PX4 in addition to automatic takeoff and landing.

Several other open-source flight controller software worth mentioning are OpenPilot [24], LibrePilot [17], BetaFlight [44], dRonin [12], and INAV [16].

1.6.3 Computational Unit

The computational resources on the UAV is a primary concern when it comes to deploying deep learning solutions. The payload capacity of the UAV and the power consumption of the processors are the two major determinants for onboard UAV processor selection. Further, given two processor platforms of comparable weight, an essential performance metric for selection could be the ratio of the inference speed

of the deep learning solution to the power consumption of the processor. Additional metrics for selection could be the memory space and volume of the processors.

There are several computational platforms such as Raspberry Pi 4 Model B, Odroid XU4, Jetson Tegra K1, Snapdragon flight board, Jetson TX1, among others with on-chip Central Processing Units (CPUs) and GPUs. Table 1.4 shows a comparison of these platforms in terms of various metrics such as memory, CPU, CPU speed, GPU, GPU performance, and dimensions.

Raspberry Pi 4 Model B (Pi 4B) is a small, low-cost 1.5GHz 64-bit ARM Cortex-A72 CPU-based hardware platform with multiple Random Access Memory (RAM) options developed for educational purpose. The Pi is also equipped with a Broadcom VideoCore VI GPU. However, the Pi 4 model B has a very high power draw in contrast to its predecessors.

Odroid XU4 is a developmental platform that is based on Samsung Exynos 5422 Octa-core CPU and ARM Mali-T628 6 Core GPU. The XU4 consists of two sockets with 1.4GHz ARM Cortex-A7 and 2GHz ARM Cortex-A15 CPUs. The Mali-T628 supports OpenGL ES 3.1/2.0/1.1 [122] and OpenCL 1.2 [127] full profile.

Jetson Tegra K1 (TK1) is a developmental kit from NVIDIA comprising of Kepler GPU with 192 CUDA cores and 4-Plus-1 quad-core ARM Cortex-A15 CPU. The TK1 has a very low power footprint while being capable of 300 GigaFloating Point Operations Per Second (FLOPS) of 32-bit floating-point computations. The Jetson TX1 on the other hand hosts an NVIDIA Maxwell 256 CUDA core GPU and quad-core ARM Cortex-A57 CPU. The power draw for a typical CUDA load is in the range of 8-10W. In contrast to TK1, the TX1 comes at a much lower form factor of 50mm × 80mm.

The Snapdragon flight board based on Snapdragon 801 processor was introduced by Qualcomm for autonomous vehicle platforms. The board comes with a 2.26GHz Qualcomm Krait quad-core CPU and Qualcomm Adreno 330 GPU with nearly 148 GigaFLOPS and 4GB RAM. In contrast to TX1 and TK1, the Snapdragon flight board comes at a smaller form factor of 58mm × 40mm. Such a smaller form factor (nearly half the size of a credit card) and lightweight (<13g) would serve as an ideal payload option for UAVs.

Here, we briefly discussed a few computational platforms that could potentially enable deep learning solutions on UAV platforms and contrasted them on the basis of their physical and performance specifications. Next, we will discuss the UAS safety and regulations enforced to prevent risk and/or injury to people and property.

1.6.4 UAS Safety and Regulations

Safety

UAVs have become increasingly popular recently for a diverse array of applications including but not limited to personal hobby, photography, aerial survey, precision agriculture, power-line inspection, entertainment, tactical surveillance, border secu-

Table 1.4 Computational Platforms for UAV

Platforms	CPU	GPU	Dimensions	Memory
Pi 4B	ARM Cortex-A72 Speed: 1.5GHz	Broadcom VideoCore VI 32GigaFLOPS	85mm×56mm	RAM Options: 2GB, 4GB, 8GB
Odroid XU4	ARM Cortex-A7 Speed: 1.4GHz ARM Cortex-A15 Speed: 2GHz	ARM Mali-T628 102.4GigaFLOPS	83mm×59mm	2GB RAM eMMC5.0 HS400 Flash
Jetson TK1	ARM Cortex-A15 Speed: 2.3GHz	Kepler 192 CUDA core 300GigaFLOPS	127mm×127mm	2GB RAM 16GB Flash
Jetson TX1	ARM Cortex-A57 Speed: 2GHz	Maxwell 256 CUDA core 1TeraFLOPS	50mm×87mm	4GB RAM 16GB Flash
Snapdragon Flight	Qualcomm Krait 400 Speed: 2.26GHz	Qualcomm Adreno 330 148GigaFLOPS	58mm×40mm	2GB RAM 32GB Flash

riety, etc. Federal Aviation Administration (FAA) estimates an even increased adoption of UAVs in the coming years with an estimate of nearly 3.5 Million in 2021 [43]. The advent of UAVs have posed significant safety and security challenges. Safety encompasses physical risks posed to people and infrastructure as well as UAV cyber-security risks. FAA has reported over 4889 incidents causing serious harm to people and infrastructure between 2014 and 2017 [33]. UAV risk factors such as obstacle collision, human factor, rogue UAVs, untimely battery, and sensor errors, etc., must be carefully assessed prior to any UAV missions. Such risk assessment becomes increasingly essential when opting for self-designed UAVs as opposed to commercial drones. As discussed in section 1.6.2, most of the commercial drones incorporate general safety measures as part of the flight controller software such as obstacle avoidance, return home or land when battery low or sensor error, geofence, among others. Hence, strict UAV safety assessment must be conducted in a studied and regulatory manner to alleviate risks to the mission as well as people and infrastructure.

Regulations

In the United States, FAA is the regulatory body that enforces aviation rules for air traffic control. Commercial as well as hobbyist use of UAVs must abide by the regulations enforced by FAA as detailed in [38]. The rules and regulations are enforced based on weight, coverage distance, application, speed, and flight altitude. The regulations restrict operating UAVs over/near people, in certain airspaces (airports, military facilities, or no-fly zones), and non-line-of-sight operation to avoid accidents and injuries. Commercial UAV operation requires the pilots to get licenses as well

as are restricted to operate during daylight hours. Recreational flying involves similar rules such as registering the UAV, line-of-sight operation, daylight operation, drone altitude not more than 400 feet from the ground, restricted from operating near manned aircraft, people, automobiles, and mental as well as physical alertness during drone operation.

1.7 Conclusion

This chapter presented how the modern era of machine learning can overcome challenges and accelerate the realization of truly autonomous UAS. We begin by presenting a tutorial study of the basic deep learning and reinforcement learning techniques to refine the reader's perception and equip them for further research in this realm. Next, the recent advances in deep learning and reinforcement learning techniques as applied to various autonomous UAV tasks were reviewed in depth. The inherent challenges and open problems pertaining to the application of machine learning techniques for autonomous UAS tasks were clearly stated to open doors for future research. Additionally, to bridge the gap between simulations and hardware implementations, we present a detailed account of the various simulation suites, UAV platforms, flight stacks, and regulatory standards. The various challenges and factors to consider while prototyping UAV for machine learning solutions were also discussed. Furthermore, this chapter will serve as a comprehensive handbook to pave a clear roadmap for future research and development in pursuing autonomous UAS solutions.

List of Acronyms

AI	Artificial Intelligence
AGL	Above Ground Level
ANN	Artificial Neural Network
CPU	Central Processing Unit
COTS	commercial-off-the-shelf
CAGR	Compound Annual Growth Rate
CNN	Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
D3QN	Dueling Double Deep Q-Network
DART	Dynamic Animation and Robotics Toolkit
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
DoD	Department of Defense
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EKF	Extended Kalman Filter
ESC	Event Sound Classification
FALA	Finite Action-set Learning Automata
FLOPS	Floating Point Operations Per Second
FVI	Fitted Value Iteration
GPU	Graphics Processing Unit
GCS	Ground Control Station
GMM	Gaussian Mixture Model
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IR	Infrared
ISR	Intelligence, Surveillance and Reconnaissance
LSTM	Long Short Term Memory
LWLR	Locally Weighted Linear Regression
MBRL	Model Based Reinforcement Learning

MSL Mean Sea Level
MAV Micro Aerial Vehicle
MDP Markov Decision Process
MPC Model Predictive Control
NUI Natural User Interface
ODE Open Dynamics Engine
OGRE Object-Oriented Graphics Rendering Engine
PPM Pulse Position Modulation
PPO Proximal Policy Optimization
PID Proportional-Integral-Derivative
PS-DNN Partially Shared-Deep Neural Network
RAM Random Access Memory
RDPG Recurrent Deterministic Policy Gradient
RNN Recurrent Neural Network
RL Reinforcement Learning
ROS Robot Operating System
SARSA State-action-reward-state-action
SDF Simulation Description Format
SLAM Simultaneous Localization and Mapping
STARMAC Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control
STFT Short Time Fourier Transform
SGD Stochastic Gradient Descent
SWaP Size, Weight, and Power
TD Temporal Difference
TRPO Trust Region Policy Optimization
UAS Unmanned Aerial System
UAV Unmanned Aerial Vehicle
UCT Upper Confidence bounds applied to Trees
VTOL Vertical Takeoff and Landing
SDK Software Development Kit
API Application Programming Interface
RTF Ready-to-Fly
FAA Federal Aviation Administration
AMA Academy of Model Aeronautics
GPL GNU General Public License
BSD Berkeley Software Distribution
MAVLink Micro Air Vehicle Communication Protocol
GUI Graphical User Interface

References

1. Department of Defense (DoD) (2010). U.S. Army —Unmanned Aircraft Systems Roadmap 2010-2035. URL <https://fas.org/irp/program/collect/uas-army.pdf>. Accessed: June. 27, 2020
2. Tech Talk – Unraveling 5 Levels of Drone Autonomy. URL <https://dronelife.com/2019/03/11/droneii-tech-talk-unraveling-5-levels-of-drone-autonomy/>. Accessed: June. 27, 2020
3. Aerostack. URL <https://github.com/Vision4UAV/Aerostack>. Accessed: July. 18, 2020
4. AirSim. URL <https://github.com/Microsoft/AirSim>. Accessed: July. 15, 2020
5. Airsim image. URL <https://www.microsoft.com/en-us/research/wp-content/uploads/2017/02/AirSimDemo-480x280.png>
6. Ardupilot. URL <https://ardupilot.org/>
7. Dji inspire 2. URL <https://www.dji.com/inspire-2>
8. Dji matrice 100. URL <https://www.dji.com/matrice100/info>
9. Dji matrice 600 pro. URL <https://www.dji.com/matrice600-pro>
10. Dji phantom 4 pro v2.0. URL <https://www.dji.com/phantom-4-pro-v2/>
11. Dronecode, dronecode project. URL <https://www.dronecode.org/>
12. dronin documentation. URL <http://dronin.org/docs/>
13. Gazebo. URL <http://gazebo.org/>. Accessed: July. 15, 2020
14. Gazebo simulation image. URL <https://dev.px4.io/v1.9.0/assets/simulation/gazebo.png>
15. hector_quadrotor - ROS Wiki. URL http://wiki.ros.org/hector_quadrotor. Accessed: July. 15, 2020
16. Inav source code. URL <https://github.com/iNavFlight/inav/>
17. Librepilot team, librepilot project. URL <http://librepilot.org/>
18. Matrice 200 series v2. URL <https://www.dji.com/matrice-200-series-v2>
19. Matrice 300 rtk. URL <https://www.dji.com/matrice-300>
20. Mavic 2. URL <https://www.dji.com/mavic-2>
21. Mavic 2 enterprise. URL <https://www.dji.com/mavic-2-enterprise?site=developer>
22. OGRE - Open Source 3D Graphics Engine — Home of a marvelous rendering engine. URL <https://www.ogre3d.org/>. Accessed: July. 15, 2020
23. Open source autopilot for drones - px4 autopilot. URL <https://px4.io/>
24. Openpilot, openpilot source code. URL <https://github.com/openpilot/OpenPilot>
25. P4 multispectral. URL <https://www.dji.com/p4-multispectral/>
26. Paparazzi. URL <https://wiki.paparazziuav.org/>
27. Phantom 4 rtk. URL <https://www.dji.com/phantom-4-rtk>
28. rotors_simulator - ROS Wiki. URL http://wiki.ros.org/rotors_simulator. Accessed: July. 15, 2020
29. Ryze tello. URL <https://www.ryzerobotics.com/tello-edu>
30. Unmanned Aerial Vehicle (UAV) Market. URL <https://www.marketsandmarkets.com/Market-Reports/unmanned-aerial-vehicles-uav-market-662.html>. Accessed: June. 27, 2020
31. Vtol uav with the cruise efficiency of a conventional fixed wing uav. URL <https://technology.nasa.gov/patent/LAR-TOPS-241>
32. Mq-9 reaper (2015). URL <https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104470/mq-9-reaper/>
33. Civilian drone safety incidents keep rising (2017). URL <https://www.insurancejournal.com/news/national/2017/12/08/473529.htm>
34. Anafi thermal (2020). URL <https://www.parrot.com/business-solutions-us/parrot-professional/anafi-thermal>
35. Anafi usa (2020). URL <https://www.parrot.com/us/drones/anafi-usa>
36. Anafi work (2020). URL <https://www.parrot.com/business-solutions-us/parrot-professional/anafi-work>
37. Abbeel, P., Coates, A., Quigley, M., Ng, A.: An application of reinforcement learning to aerobatic helicopter flight. pp. 1–8 (2006)
38. Administration, F.A.: U.S. Department of Transportation. URL <https://faadronezone.faa.gov/>

39. Afouras, T., Chung, J.S., Senior, A., Vinyals, O., Zisserman, A.: Deep audio-visual speech recognition. *IEEE Transaction on Pattern Analysis and Machine Intelligence* pp. 1–1 (2018)
40. Al-Emadi, S., Al-Ali, A., Mohammad, A., Al-Ali, A.: Audio based drone detection and identification using deep learning. In: *Proc. of International Wireless Communications Mobile Computing Conference (IWCMC)*, pp. 459–464 (2019)
41. Arik, S.Ö., Kliegl, M., Child, R., Hestness, J., Gibiansky, A., Fougner, C., Prenger, R., Coates, A.: Convolutional recurrent neural networks for small-footprint keyword spotting. *ArXiv abs/1703.05390* (2017)
42. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* **34**(6), 26–38 (2017). DOI 10.1109/msp.2017.2743240. URL <http://dx.doi.org/10.1109/MSP.2017.2743240>
43. Atkinson, W.: Drones are gaining popularity (2018). URL <https://www.ecmag.com/section/your-business/drones-are-gaining-popularity>
44. B, B.: Betaflight source code. URL <https://github.com/betaflight/betaflight/>
45. Bagnell, J.A., Schneider, J.G.: Autonomous helicopter control using reinforcement learning policy search methods. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 2, pp. 1615–1620 vol.2 (2001)
46. Benjdira, B., Khurshed, T., Koubaa, A., Ammar, A., Ouni, K.: Car detection using unmanned aerial vehicles: Comparison between faster r-cnn and yolov3 (2018)
47. Bou-Ammar, H., Voos, H., Ertel, W.: Controller design for quadrotor uavs using reinforcement learning. In: *2010 IEEE International Conference on Control Applications*, pp. 2130–2135 (2010)
48. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: *Openai gym* (2016)
49. Bøhn, E., Coates, E.M., Moe, S., Johansen, T.A.: Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In: *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 523–533 (2019)
50. Caleberg: Open source drones: An intro to the ardupilot flight stack (2019) (2019). URL <https://dojofordrones.com/ardupilot-flight-stack/>
51. Carrio, A., Sampedro, C., Rodriguez-Ramos, A., Campoy, P.: A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors* **2017** (2017)
52. Chapman, A.: Types of drones: Multi-rotor vs fixed-wing vs single rotor vs hybrid vtol (2019). URL <https://www.auav.com.au/articles/drone-types/>
53. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: cudnn: Efficient primitives for deep learning (2014)
54. Chopra, S., Hadsell, R., LeCun, Y.: Learning a similarity metric discriminatively, with application to face verification. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 539–546 vol. 1 (2005)
55. Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: *Proc. of the International Conference on Machine Learning, ICML '08*, p. 160–167. Association for Computing Machinery, New York, NY, USA (2008). DOI 10.1145/1390156.1390177. URL <https://doi.org/10.1145/1390156.1390177>
56. Delmerico, J., Mueggler, E., Nitsch, J., Scaramuzza, D.: Active autonomous aerial exploration for ground robot path planning. *IEEE Robotics and Automation Letters* **2**(2), 664–671 (2017)
57. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09* (2009)
58. dos Santos, S.R.B., Nascimento, C.L., Givigi, S.N.: Design of attitude and path tracking controllers for quad-rotor robots using reinforcement learning. In: *2012 IEEE Aerospace Conference*, pp. 1–16 (2012)
59. Ebeid, E.S.M., Skriver, M., Terkildsen, K., Jensen, K., Schultz, U.: A survey of open-source uav flight controllers and flight simulators. *Microprocessors and Microsystems* **61** (2018). DOI 10.1016/j.micpro.2018.05.002

60. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proc. of the International Conference on Machine Learning - Volume 70, ICML'17, p. 1126–1135. JMLR.org (2017)
61. Furrer, F., Burri, M., Achtelik, M., Siegwart, R.: Robot Operating System (ROS): The Complete Reference (Volume 1), chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-26054-9_23. URL http://dx.doi.org/10.1007/978-3-319-26054-9_23
62. Gandhi, D., Pinto, L., Gupta, A.: Learning to fly by crashing. In: Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3948–3955 (2017)
63. Giusti, A., Guzzi, J., Cireşan, D.C., He, F., Rodríguez, J.P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Caro, G.D., Scaramuzza, D., Gambardella, L.M.: A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters* **1**(2), 661–667 (2016)
64. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). <http://www.deeplearningbook.org>
65. Gupta, S.G., Ghonge, D., Jawandhiya, P.M., et al.: Review of unmanned aircraft system (uas). *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 2* (2013)
66. Hausknecht, M., Stone, P.: Deep recurrent q-learning for partially observable mdps (2015)
67. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 770–778 (2016)
68. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
69. Heess, N., Hunt, J.J., Lillicrap, T.P., Silver, D.: Memory-based control with recurrent neural networks (2015)
70. Hentati, A.I., Krichen, L., Fourati, M., Fourati, L.C.: Simulation tools, environments and frameworks for uav systems performance analysis. In: 2018 14th International Wireless Communications Mobile Computing Conference (IWCMC), pp. 1495–1500 (2018)
71. Hester, T., Stone, P.: Texplora: Real-time sample-efficient reinforcement learning for robots. *Machine Learning* **90** (2013). DOI 10.1007/s10994-012-5322-7
72. Hwangbo, J., Sa, I., Siegwart, R., Hutter, M.: Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters* **2**(4), 2096–2103 (2017)
73. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 1mb model size. *ArXiv abs/1602.07360* (2017)
74. Imanberdiyev, N., Fu, C., Kayacan, E., Chen, I.: Autonomous navigation of uav by using real-time model-based reinforcement learning. In: 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 1–6 (2016)
75. Jagannath, A., Jagannath, J., Melodia, T.: Redefining Wireless Communication for 6G: Signal Processing Meets Deep Learning. arXiv preprint arXiv:2004.10715 (2020)
76. Jagannath, J., Polosky, N., Jagannath, A., Restuccia, F., Melodia, T.: Machine learning for wireless communications in the internet of things: A comprehensive survey. *Ad Hoc Networks (Elsevier)* **93**, 101913 (2019)
77. Jagannath, J., Polosky, N., Jagannath, A., Restuccia, F., Melodia, T.: Neural Networks for Signal Intelligence: Theory and Practice. In: *Machine Learning for Future Wireless Communications*, pp. 243–264. John Wiley & Sons, Limited (2020)
78. Jeon, S., Shin, J., Lee, Y., Kim, W., Kwon, Y., Yang, H.: Empirical study of drone sound detection in real-life environment with deep neural networks. In: Proc. of European Signal Processing Conference (EUSIPCO), pp. 1858–1862 (2017)
79. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proc. of the 22nd ACM International Conference on Multimedia, MM '14, p. 675–678. Association for Computing Machinery, New York, NY, USA (2014). DOI 10.1145/2647868.2654889. URL <https://doi.org/10.1145/2647868.2654889>

80. Kim, D.K., Chen, T.: Deep neural network for real-time autonomous indoor navigation. *CoRR* **abs/1511.04668** (2015). URL <http://arxiv.org/abs/1511.04668>
81. Kim, H.J., Jordan, M.I., Sastry, S., Ng, A.Y.: Autonomous helicopter flight via reinforcement learning. In: S. Thrun, L.K. Saul, B. Schölkopf (eds.) *Advances in Neural Information Processing Systems 16*, pp. 799–806. MIT Press (2004). URL <http://papers.nips.cc/paper/2455-autonomous-helicopter-flight-via-reinforcement-learning.pdf>
82. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *ArXiv* **abs/1412.6980** (2014)
83. Kober, J., Bagnell, J., Peters, J.: Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* **32**, 1238–1274 (2013). DOI 10.1177/0278364913495721
84. Koch, W., Mancuso, R., Bestavros, A.: Neuroflight: Next generation flight control firmware. *arXiv preprint arXiv:1901.06553* (2019)
85. Koch, W., Mancuso, R., West, R., Bestavros, A.: Reinforcement learning for uav attitude control. *ACM Trans. Cyber-Phys. Syst.* **3**(2) (2019). DOI 10.1145/3301273. URL <https://doi.org/10.1145/3301273>
86. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE Cat. No.04CH37566), vol. 3, pp. 2149–2154 vol.3 (2004)
87. Kouris, A., Bouganis, C.: Learning to fly by myself: A self-supervised cnn-based approach for autonomous navigation. In: *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9 (2018)
88. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: *Proc. of the 25th International Conference on Neural Information Processing Systems - Volume 1, NIPS'12*, p. 1097–1105. Red Hook, NY, USA (2012)
89. Kuzovkin, I., Vicente, R., Petton, M., Lachaux, J., Baciú, M., Kahane, P., Rheims, S., Vidal, J.R., Aru, J.: Activations of deep convolutional neural networks are aligned with gamma band activity of human visual cortex. *Communications Biology* **1** (2018)
90. Lambert, N.O., Drew, D.S., Yaconelli, J., Levine, S., Calandra, R., Pister, K.S.J.: Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters* **4**(4), 4224–4230 (2019)
91. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*, pp. 2278–2324 (1998)
92. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: *Proc. of IEEE International Symposium on Circuits and Systems*, pp. 253–256 (2010)
93. Li, W., Fu, H., Yu, L., Cracknell, A.: Deep learning based oil palm tree detection and counting for high-resolution remote sensing images. *Remote Sensing* **9**(1), 22 (2016). DOI 10.3390/rs9010022. URL <http://dx.doi.org/10.3390/rs9010022>
94. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2015)
95. Lin, T., Yin Cui, Belongie, S., Hays, J.: Learning deep representations for ground-to-aerial geolocalization. In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5007–5015 (2015)
96. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., Van Der Laak, J.A., Van Ginneken, B., Sánchez, C.I.: A survey on deep learning in medical image analysis. *Medical image analysis* **42**, 60–88 (2017)
97. Liu, Y., Cen, C., Che, Y., Ke, R., Ma, Y., Ma, Y.: Detection of maize tassels from uav rgb imagery with faster r-cnn. *Remote Sensing* **12**(2), 338 (2020). DOI 10.3390/rs12020338. URL <http://dx.doi.org/10.3390/rs12020338>
98. Madaan, R., Gyde, N., Vemprala, S., Brown, M., Nagami, K., Taubner, T., Cristofalo, E., Scaramuzza, D., Schwager, M., Kapoor, A.: Airsim drone racing lab. *arXiv preprint arXiv:2003.05654* (2020)
99. Mairaj, A., Baba, A.I., Javaid, A.Y.: Application specific drone simulators: Recent advances and challenges. *Simulation Modelling Practice and Theory* **94**, 100–117 (2019). DOI 10.1016/j.simpat.2019.01.004. URL <http://dx.doi.org/10.1016/j.simpat.2019.01.004>

100. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., Von Stryk, O.: Comprehensive simulation of quadrotor uavs using ros and gazebo. pp. 400–411 (2012). DOI 10.1007/978-3-642-34327-8_36
101. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning (2013)
102. Morito, T., Sugiyama, O., Kojima, R., Nakadai, K.: Partially shared deep neural network in sound source separation and identification using a uav-embedded microphone array. In: Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1299–1304 (2016)
103. Nasse, F., Thureau, C., Fink, G.A.: Face detection using gpu-based convolutional neural networks. In: Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns, CAIP '09, p. 83–90. Springer-Verlag, Berlin, Heidelberg (2009)
104. Ng, A., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.: Inverted autonomous helicopter flight via reinforcement learning. Proceedings of the International Symposium on Experimental Robotics (2004)
105. Otter, D.W., Medina, J.R., Kalita, J.K.: A survey of the usages of deep learning for natural language processing. IEEE Transactions on Neural Networks and Learning Systems pp. 1–21 (2020)
106. Pham, H.X., La, H.M., Feil-Seifer, D., Nguyen, L.V.: Autonomous uav navigation using reinforcement learning (2018)
107. Pham, H.X., La, H.M., Feil-Seifer, D., Van Nguyen, L.: Reinforcement learning for autonomous uav navigation using function approximation. In: 2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), pp. 1–6 (2018)
108. Polvara, R., Patacchiola, M., Sharma, S., Wan, J., Manning, A., Sutton, R., Cangelosi, A.: Toward end-to-end control for uav autonomous landing via deep reinforcement learning. In: 2018 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 115–123 (2018)
109. Polydoros, A.S., Nalpantidis, L.: Survey of model-based reinforcement learning: Applications on robotics. Journal of Intelligent & Robotic Systems **86**(2), 153–173 (2017)
110. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788 (2016)
111. Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6517–6525 (2017)
112. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. ArXiv **abs/1804.02767** (2018)
113. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1137–1149 (2017). DOI 10.1109/TPAMI.2016.2577031. URL <https://doi.org/10.1109/TPAMI.2016.2577031>
114. Rodríguez Ramos, A., Sampedro Pérez, C., Bavle, H., de la Puente, P., Campoy, P.: A deep reinforcement learning strategy for uav autonomous landing on a moving platform. Journal of Intelligent & Robotic Systems (2018). DOI 10.1007/s10846-018-0891-8
115. Rumelhart, D., Smolensky, P., McClelland, J.L., Hinton, G.E.: Schemata and sequential thought processes in pdp models (1986)
116. Sanchez-Lopez, J., Pestana, J., de la Puente, P., Campoy, P.: A reliable open-source system architecture for the fast designing and prototyping of autonomous multi-uav systems: Simulation and experimentation. Journal of Intelligent & Robotic Systems (2015). DOI 10.1007/s10846-015-0288-x
117. Sanchez-Lopez, J.L., Suárez Fernández, R.A., Bavle, H., Sampedro, C., Molina, M., Pestana, J., Campoy, P.: Aerostack: An architecture and open-source software framework for aerial robotics. In: 2016 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 332–341 (2016)
118. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization (2015)
119. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017)

120. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* **45**(11), 2673–2681 (1997)
121. Sebe, N., Cohen, I., Garg, A., Huang, T.S.: *Machine learning in computer vision*, vol. 29. Springer Science & Business Media (2005)
122. Segal, M., Akeley, K.: The opengl graphics system: a specification (version 1 (2003)
123. Shah, S., Dey, D., Lovett, C., Kapoor, A.: Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: *Field and Service Robotics* (2017). URL <https://arxiv.org/abs/1705.05065>
124. s.han, A.: A guide to the different types of drones & uas (2018). URL <https://www.evergladesuniversity.edu/guide-different-types-drones-unmanned-aerial-systems/>
125. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: *International Conference on Learning Representations* (2015)
126. Singla, A., Padakandla, S., Bhatnagar, S.: Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge. *IEEE Transactions on Intelligent Transportation Systems* pp. 1–12 (2019)
127. Stone, J.E., Gohara, D., Shi, G.: Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in Science Engineering* **12**(3), 66–73 (2010)
128. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*, 2 edn. (2018)
129. Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9 (2015)
130. Tsitsiklis, J.N., Van Roy, B.: An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control* **42**(5), 674–690 (1997)
131. Wang, C., Wang, J., Shen, Y., Zhang, X.: Autonomous navigation of uavs in large-scale complex environments: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology* **68**(3), 2124–2136 (2019)
132. Wang, Y., Sun, J., He, H., Sun, C.: Deterministic policy gradient with integral compensator for robust quadrotor control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* pp. 1–13 (2019)
133. Waslander, S.L., Hoffmann, G.M., Jung Soon Jang, Tomlin, C.J.: Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3712–3717 (2005)
134. Xu, Y., Yu, G., Wang, Y., Wu, X., Ma, Y.: Car detection from low-altitude uav imagery with the faster r-cnn. *Journal of Advanced Transportation* **2017**, 1–10 (2017). DOI 10.1155/2017/2823617
135. Zhang, T., Kahn, G., Levine, S., Abbeel, P.: Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 528–535 (2016)
136. Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., Oliva, A.: Learning deep features for scene recognition using places database. In: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (eds.) *Advances in Neural Information Processing Systems* 27, pp. 487–495. Curran Associates, Inc. (2014). URL <http://papers.nips.cc/paper/5349-learning-deep-features-for-scene-recognition-using-places-database.pdf>
137. Zou, Q., Wang, Y., Wang, Q., Zhao, Y., Li, Q.: Deep learning-based gait recognition using smartphones in the wild. *IEEE Transactions on Information Forensics and Security* **15**, 3197–3212 (2020)