Low-Level control of small scale helicopter using Soft Actor-Critic method

by

Majid Kamyab

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Chemical Engineering

Department of Chemical Engineering

University of Alberta

Abstract

Unmanned Aerial Vehicles (UAVs), or drones, have been employed in a variety of applications, ranging from surveillance to emergency operations. These systems comprise an "inner loop" that provides stability and control and an "outer loop" in charge of mission-level tasks, such as waypoint navigation. Despite their inherent instability, different techniques for controlling these robots have been devised under stable environmental conditions. However, these algorithms must know a robot's dynamics to be effective; furthermore, more complex control is necessary for UAVs to perform in unstable environmental conditions. In this research, a simulated drone has been successfully controlled using model-free reinforcement learning with no prior knowledge of the robot's model. Soft Actor-Critic (SAC) method is trained to perform low-level control of a small-scaled helicopter in a set-point control system. First, a simulation environment is created in which all tests were carried out and then it is shown that SAC can not only develop a strong policy, but it can also deal with unknown circumstances.

Acknowledgements

I want to express my gratitude to my distinguished supervisor, Dr. Dubljevic, for his essential supervision, support, and instruction throughout my Master's degree. My thanks go to the Faculty of Engineering for providing me with the money to pursue my studies at the University of Alberta's Department of Chemical Engineering. In addition, I'd like to thank Dr. Koch for his invaluable assistance. I'd want to thank my friends, lab mates, colleagues, and research team – Hamid Khatibi – for a memorable time spent together in academic and social contexts. My thanks also go to my mother, brother, and father for their support during my education.

Contents

List of Tables

List of Figures

Nomenclature

Subscripts

cg	Center of gravity	$[-]$
d	Desired value for the variable	$[-]$
fus	Fuselage	$[-]$
ht	Horizontal tail	$[-]$
mr	Main rotor	$[-]$
tr	Tail rotor	$[-]$
vt	Vertical tail	$[-]$

Symbols

$\mathbb{E}[X]$	Expected value of a random variable X	
α_1	Stabilizer bar rate derivative	$53\,[-]$
α_2	Stabilizer bar input derivative	$55\,[-]$
α_{tail}	Slope of the tail servo angle to the PW of the signal	$-1698.5\,[rad/s]$
α_{tail}	Slope of the tail servo angle to the PW of the signal	$[rad/s]$
$\delta_{0_{tail}}$	Y-intercept of the tail servo angle to the PW of the signal	$[rad]$
$\delta_{0_{tail}}$	Y-intercept of the tail servo angle to the PW of the signal	$1.4724\,[rad]$
δ_{coll}	Main rotor collective pitch input	$[rad]$

δ_{lat}	Lateral cyclic pitch input	$[rad]$
δ_{lon}	Longitudinal cyclic pitch input	$[rad]$
δ_{ped}	Tail rotor blade pitch input	$[rad]$
δ_x, δ_y	Euler rotation angles of the swashplate	$[rad]$
η	Sliding surface reach time factor	$[-]$
γ	Discounted rate in RL.	$[rad]$
λ	surface convergence rate factor	$[-]$
λ_0	Main rotor inflow ratio	$[-]$
$\mathbb{V}$	State value	$[-]$
$\mathcal{D}$	Buffer in RL.	$[-]$
μ	Advance ratio	$[-]$
μ_x	Non-dimensional airflow components along x axis	$[-]$
μ_y	Non-dimensional airflow components along y axis	$[-]$
μ_z	Non-dimensional airflow components along z axis	$[-]$
Ω	Nominal main rotor speed	$115\,[rad/s]$
$\overline{\mu}$	Normalized Advance ratio	$[-]$
Φ	Angular velocity transformation matrices from the body to inertial coordinates	$[-]$
ϕ, θ, ψ	Euler angles	$[rad]$
Ψ	Blade azimuth angle	$[rad]$
ρ	Air density	$1.107\,[kg/m^3]$
σ_{mr}	Main rotor solidity factor	$[-]$

τ_f	Main rotor flapping time-constant	$0.04\,[s]$
τ_{mr}	Main rotor blade element radial distance ratio	$[-]$
τ_s	Stabilizer bar flapping time-constant	$0.2\,[s]$
τ_{tr}	Tail rotor blade element radial distance ratio	$[-]$
$\Theta_{0_{tail}}$	Zero pitch angle of the tail blade	$[rad]$
$\theta_{0_{tail}}$	Zero pitch angle of the tail blade	$0.1169\,[rad]$
Θ_{mr}	Pitch angle of the main rotor	$[rad]$
Θ_{tr}	Pitch angle of the tail rotor	$[rad]$
ξ	Ranodm number generated in a normal distribution.	$[-]$
a_1	Coefficient of the first harmonic approximation in the Fourier series representation of the rotor flapping equations in x direction	$[rad/s]$
a_κ	Tunable parameter in policy search of RL.	$[-]$
a_μ	Mean (expectation) of action in a gaussian distribution.	$[-]$
a_σ	Standard deviation of action in a gaussian distribution.	$[-]$
A_b	Lateral flapping cross-coupling derivative	$-0.1\,[-]$
a_t	Action of the agent in RL	$[-]$
a_v	Longitudinal translational velocity contributions to the flapping of the main rotor	$[rad/s]$
A_{lon}	Longitudinal cyclic to flap gain at nominal rpm	$1\,[-]$
B	Body coordinates	
b_1	Coefficient of the first harmonic approximation in the Fourier series representation of the rotor flapping equations in y direction	$[rad/s]$
B_A	Longitudinal flapping cross-coupling derivative	$0.1\,[-]$

B_e	bound on b	$[-]$
b_v	Lateral translational velocity contributions to the flapping of the main rotor	$[rad/s]$
B_{lat}	Lateral cyclic to flap gain at nominal rpm	$0.9875\,[-]$
$C_{D0_{tr}}$	Tail rotor zero lift drag coefficient	$0.06\,[-]$
C_{D0}	Main rotor blade zero lift drag coefficient	$0.01\,[-]$
C_{L0}	Main rotor blade zero lift curve slope	$0.008\,[-]$
$C_{L\alpha_{tr}}$	Tail rotor blade lift curve slope	$4.95\,[rad^{-1}]$
$C_{L\alpha}$	Main rotor blade lift curve slope	$5.49\,[rad^{-1}]$
c_{mr}	Main rotor chord	$0.082\,[m]$
c_{tr}	Tail rotor chord	$0.025\,[m]$
d	a boolean indicating wheter it is a terminal state or not	$[-]$
F	Vectors of external forces	$[kg.m/s^2]$
F_e	bound on f	$[-]$
F_x	Force along x axis	$[kgm/s^2]$
F_y	Force along y axis	$[kgm/s^2]$
F_z	Force along z axis	$[kgm/s^2]$
G	Expected return of MDP process.	$[-]$
H	Entropy of a stochastic policy.	$[-]$
I	Inertia coordinate	
I_s	Equivalent moment of inertia tensor of the TPP rotor disk	
I_{xx}	Rolling moment of inertia	$0.3\,[kg.m^2]$

I_{yy}	Pitching moment of inertia	$1.6\,[kg.m^2]$
I_{zz}	Yawing moment of inertia	$2.0\,[kg.m^2]$
J	Jacobian matrix	$[-]$
K_λ	Main rotor downwash factor at fuselage	$1\,[-]$
K_c	Longitudinal flapping due to the stabilizier bar factor	$[-]$
K_d	Lateral flapping due to the stabilizier bar factor	$[-]$
K_s	Longitudinal flapping cross-coupling derivative	$0.3\,[-]$
K_u	Flapping due to the forward velocity factor	$[-]$
K_v	Flapping due to the sideway velocity factor	$[-]$
K_β	Hub torsional stiffness	$255\,[N.m]$
K_μ	Scaling of flap response to speed variation	$[-]$
K_{lat}	Lateral cyclic to lateral flap gain	$0.98\,[-]$
K_{lon}	Longitudinal cyclic to longitudinal flap gain	$1\,[-]$
L	Mean Squared Bellman Error MSBE.	$[-]$
l	learning rate	$0.3\,[kg.m^2]$
M	Vectors of external moments	$[kg.m^2/s^2]$
m	Helicopter mass	$11.5\,[kg]$
M_x	Moment along x axis	$[kgm^2/s^2]$
M_y	Moment along y axis	$[kgm^2/s^2]$
M_z	Moment along y axis	$[kgm^2/s^2]$
n_{tr}	Gear ratio of tail rotor to main rotor	$6\,[-]$

p	Angular rate component (pitch)along x-axis of the CG in I rotated into B	$[rad/s]$
q	Angular rate component (roll) along y-axis of the CG in I rotated into B	$[rad/s]$
Q_{mr}	Drag torque of main rotor	$kg.m/s^2$
r	Angular rate component (yaw) along z-axis of the CG in I rotated into B	$[rad/s]$
R_b^I	Linear velocity transformation matrices from the body to inertial coordinates	$[-]$
R_t	Reward of the environment in RL	$[-]$
R_{mr}	Main rotor radius	$0.95\,[m]$
R_{tr}	Tail rotor radius	$0.15\,[m]$
S_h	Horizontal tail area	$[m^2]$
s_t	Observation of the environment in RL	$[-]$
S_vt	Vertical tail area	$[m^2]$
S_x^{fus}	Frontal fuselage area	$0.1\,[m^2]$
S_y^{fus}	Side fuselage area	$0.83\,[m^2]$
S_z^{fus}	Vertical fuselage area	$0.51\,[m^2]$
T	Thrust	$kg.m/s^2$
U	Vector of input $[\delta_{col}, \delta_{lat}, \delta_{lon}, \delta_{ped}]^T$	$[-]$
u	Velocity component along x-axis of the CG in I rotated into B	$[m/s]$
u_n	Normalized air relative velocity comp along n-axis. in the main rotor rpn coord.	$[-]$

u_p	Normalized air relative velocity comp along p-axis. in the main rotor rpn coord.	$[-]$
u_r	Normalized air relative velocity comp along r-axis. in the main rotor rpn coord.	$[-]$
$u_{n_{tr}}$	Normalized air relative velocity along n-axis in the tail rotor rpn coord.	$[-]$
$u_{p_{tr}}$	Normalized air relative velocity along pr-axis in the tail rotor rpn coord.	$[-]$
$u_{r_{tr}}$	Normalized air relative velocity along r-axis in the tail rotor rpn coord.	$[-]$
u_{wind}	Wind velocity along x-axis of the CG in I rotated into B	$[m/s]$
v	Velocity component along y-axis of the CG in I rotated into B	$[m/s]$
V_a	Normal-to-the-disk component of the free stream velocity normalized by V_h	$[-]$
v_a	Axial inflow ratio	$[-]$
V_{fus}	Dynamic pressure of the fuselage.	$[m/s]$
V_h	Main rotor induced velocity in hover	$[m/s]$
$V_{i_{tr}}$	tail rotor induced velocity	$[m/s]$
V_i	Main rotor induced velocity	$[m/s]$
v_{wind}	Wind velocity along y-axis of the CG in I rotated into B	$[m/s]$
w	Velocity component along z-axis of the CG in I translated into B	$[m/s]$
w_{wind}	Wind velocity along z-axis of the CG in I rotated into B	$[m/s]$
x, y, z	Position of CG in I coordinates	$[m]$
x_{fus}	Tail rotor hub offset from CG along x-axis	$-1.22\,[m]$
x_{ht}	Horizontal tail offset from CG along x-axis	$[m]$
x_{vt}	Horizontal tail offset from CG along x-axis	$[m]$

z_{cg}	Main rotor hub height from CG	$-0.32\,[m]$
z_c	Vertical displacement of the swashplate	$[m]$
z_{vt}	Vertical tail offset from CG along z-axis	$[m]$

Acronyms

CG	Center of gravity	$[-]$

Chapter 1

Introduction

1.1 Autonomous UAV

Unmanned aerial vehicles (UAVs) are aircraft with no human on board. They are controlled remotely or automatically. Unmanned Aerial Vehicles (UAV) are gaining popularity, both in terms of academic research and potential applications [1].

Classification of the UAVs has two major sub-classes of fixed-wing and rotary-wing. the rotary-wing UAVs received growing attention in recent years thanks to the improvements in embedded microprocessors and batteries. surveillance [2, 3], disaster management [4, 5], and rescue missions [6] are only a few numbers of examples of the broad implementation field of the rotary-wing UAVs.

The majority of recent years' research is focused on quadcopters which are rotary-wing aircraft with four rotors [7, 8, 9, 10] Thanks to their agility and ease of control. On the other hand, single rotor helicopters have gotten less attention from researchers, mainly because they are intrinsically unstable; they have highly coupled nonlinear dynamics, and wind gusts can easily disturb them. These factors lead to a complex control problem for single rotor small-scaled helicopters. However, the payload capacity of these helicopters is superior to quadcopters, making them more suitable for transportation in emergency situations [11]. As single rotor small, scaled helicopters received less attention, in this study, we will focus on this type of UAVs.

As a single-rotor helicopter is unstable by nature, it requires a flight control system that operates the vehicle, which is like a human pilot in a large, scaled helicopter. As a result, the flight control can either accept remote control input from an operator or operate autonomously. Remote control of single rotor helicopters is not economically viable, so autonomous control is preferred for most commercial applications. Therefore, the autonomous control of unmanned aerial vehicles (UAVs) is the goal of this research.

1.2 Traditional Control Systems

Control of single rotor helicopters is studied through classic (continuous) or modern (digital) control approaches. Because of its highly cross-coupling nature of single rotor small scale helicopters (SRSSH), usually, a MIMO approach is implemented [12, 13]. $H\infty$ method is also used in [14, 15] using a 30-state nonlinear model by an inner loop and outer loop technique. Sliding mode controller (SMC) is also used for control of SRSSH [16].

The issue of optimal control methods is that they all necessitate knowledge of the robot's dynamics, requiring system identification and model derivation for each UAV. Depending on the task, this can become tedious, if not impossible. Notably, the final control system will be a one-of-a-kind solution to a specialized study. These strategies may be insufficient to deal with changing conditions, unanticipated events, and stochastic environments [17].

On this basis, the following question is posed: What if the vehicle teaches itself how to perform a task optimally without using a model? This leads to the next section on reinforcement learning.

1.3 The Use of Reinforcement Learning as an Optimal Control Method

Artificial intelligence (AI) has lately caused a breakthrough in various industries worldwide, ranging from engineering to medical services. Recent advancements in computer technology and data storage, along with AI's learning capacities, have propelled AI to the forefront of numerous applications, such as object recognition and natural language processing. AI is expected to contribute more than 15 trillion USD to the global economy while increasing GDP by 26% by 2030. Overall, artificial intelligence (AI) is a powerful tool that covers many aspects of nowadays scientific achievements [18].

Machine learning (ML) is arguably the most significant branch of AI. It is described as an ability in computer systems that allows them to learn without the need for continuous control over it [19]. The area of machine learning may be divided further into supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

The term "supervised learning" refers to a situation in which the "experience," or training example, provides essential information that is absent in the unknown "test examples" whereby the learned knowledge is to be implemented. An expert provides the additional information in experience. It tries to generalize across experiences and then applies this knowledge to predict labels for test examples [20]. Since the agent tries to mimic the expert, it will not wholly provide the same response as the expert. This error is called the Bayes error rate [21].

In unsupervised learning, there is no distinction between training data and test data. A typical example of such a job is grouping data collection into subgroups of related objects. Semi-supervised learning is a combination of supervised learning and unsupervised learning. During training, semi-supervised learning mixes a small quantity of labeled data with a lot of unlabeled data, which will improve learning accuracy.

Ideally, supervised learning or semi-supervised learning can completely replicate the supervisor. However, it cannot outperform the supervisor in terms of outcomes. Reinforcement learning (RL) attempts to solve this dilemma by substantial changes to the learning process. Ultimately, the objective of RL is to enable machines to outperform all existing approaches. The RL agent tries to achieve a better result than the currently feasible ones by learning the best mapping of states to actions using a reward signal as a criterion. RL methods allow a vehicle to discover an optimal behavior on its own through trial-and-error interactions with its surroundings. This is based on the commonsense idea that if an action results in a satisfactory or better situation, the tendency to perform that action in the initial situation is *reinforced*.

RL is like classical optimal control theory [22] in engineering platform. Both theorems deal with the problem of determining an input (i.e., optimal controller in control theory or optimal policy in RL) for solving the optimization problem. Furthermore, both rely on a system's notation being described by an underlying set of states, actions, and a model that captures transitions between one state and the other. So RL can tackle the same problem that optimal control does [23, 24]. However, because the agent does not have access to the state vector dynamics, the agent must learn the repercussions of its actions via trial and error while interacting with the environment.

Although there are some recent achievements on model-based RL [25], most of the RL algorithms are model-free. They attempt to control without the knowledge of a dynamic model; in other words, it only receives the current states[*] and a reward from the environment (helicopter in this case) in each step.

This framework has received much attention in recent years, with promising outcomes in a range of domains, including outperforming human specialists on Atari games [26], Go [27], and replicating complex helicopter maneuvers. [28, 29, 30] . A remarkable range of robotics challenges may be conveniently formulated as reinforcement learning problems dating back to 1992 when the OBELIX robot is trained to push objects [31]. A model-free policy gradient technique was used to teach a Zebra Zero robot arm how to perform a peg-in-hole insertion task [32].

Recently, RL-based UAV control has received a lot of interest. The initial research generated an engineered reward function. They developed a model of robot dynamics

[*]in the fully observable Markov decision process (FOMDP). In the partially observable Markov decision process, a history of states is required in each step.

through demonstration but then employed the model in simulation, leading to the simulation of robot state while using RL to optimize a NN controller for autonomous helicopter flying [33] or inverted helicopter maneuver [29]. However, defining the reward function could be an arduous task. One solution would be to utilize an expert and award the helicopter for emulating the expert's behavior. Abbeel et al. used this approach to perform aerobatic helicopter flight [28].

In recent years, deep learning has been shown to improve the RL field [34]. Deep learning relies on neural networks' powerful function approximation properties, which can automatically find compact low-dimensional representations of high-dimensional data (e.g., images). This enabled reinforcement learning methods to scale up to previously unreachable problems.

Deep reinforcement learning has also gained attention recently in UAV control, William Koch et al. [35] compared Deep Deterministic Policy Gradient (DDPG) [36], Trust Region Policy Optimization (TRPO) [37] and Proximal Policy Optimization (PPO) [38] algorithms on the Iris quadcopter and then comparing the result to a PID controller. Although TRPO and DDPG failed to reach stability, they have shown that PPO results are powerful enough to be comparable to a PID controller. Barros and Colombini [39] also proved that the Soft Actor-Critic (SAC) [40] method can perform a low-level control on a commercial quad-rotor Parrot AR Drone 2.0. However, there is still a lack of research on a small-scaled single-rotor helicopter.

1.4 Simulation Environment for RL

In RL, the amount of try and error required to learn beneficial actions is usually high. As a result, sampling the environment is the primary challenge with reinforcement learning. One way to approach this is by having parallel similar real-world environments doing the same thing [41]. However, in the case of the UAV, failure means the loss of a UAV, and hence it is costly. This problem is exacerbated by several real-world factors that make UAVs a problematic domain for RL [42]. UAVs are frequently dangerous and costly to run during the initial training such that the aircraft will fail several times until it reaches a satisfactory performance. This will need high maintenance costs in addition to the original hardware expenditures. Moreover. Robotic have continuous high-dimensional state and action spaces, and finally, it requires a fast online response. As a result, the use of a simulation environment seems necessary for the initial learning procedure of an RL algorithm.

To compensate for the expense of real-world interactions, the UAV must first learn the behavior in simulation and then transfer it to the real vehicle. Usage of a simulator provides an affordable approach in order to create samples. In a simulation, it is possible to crash the UAV as many times as needed; In addition, no safety measures must be taken for, and there would also be no lag in the process due to maintenance or any other real-world issues. Simulations are also more reproducible; For example, wind gusts are not easy to reproduce in the real world, while in simulation, the wind gust random model can be saved and reused elsewhere.

The issue with using a simulation environment is that none of them can completely capture real-world complexity. When a policy is trained in simulation, it usually is not optimal to use in the real world [43]. One possible solution would be to initially train the policy in simulation and then perform tuning in the real world [44, 45].

1.5 Thesis Objective and outline

In this research, we wish to expand on recent research in RL, especially Deep Reinforcement Learning (DRL) to control a SRSSH. More precisely, low-level control rules are learnt directly from the UAV simulation. Notably, the purpose of this thesis is only to train the DRL technique in a simulated setting, leaving future work to examine the transfer to the actual world. In the following, the outline of this thesis is included.

1.5.1 Chapter 2: Reinforcement Learning Background

A wrong choice of RL method or its hyper-parameters can be time-consuming or even impossible to reach good stability of the UAV. This is because it mainly necessitates an extensive exploration of the state-space in order to extract acceptable policies. So, in the second chapter, a review of reinforcement learning methods is discussed. By providing a mathematical framework and describing essential components, this chapter includes a formal introduction to RL. Following that, the chapter provides an overview of Value-based and policy-based methods. Finally, the chapter introduces the DRL algorithm, SAC, which will subsequently be used for UAV control.

1.5.2 Chapter 3: Simulation environment

This chapter introduces the Simulated environment used for interaction with the RL method. First, the helicopter dynamics are discussed, including the forces applied to the UAV, such as fuselage and main rotor forces. Secondly, its effect on the 6 degrees of freedom (DOF) UAV is discussed. Finally, the environment setup is discussed, including the actions and rewards in the RL platform.

1.5.3 Chapter 4: Result and discussion

This chapter contains the results of applying the RL algorithm on a simulated environment, as well as a discussion.

1.5.4 Chapter 5: Conclusion and future work

The conclusion and recommendations for future work are given in the final section of this chapter.

Chapter 2

review of Reinforcement Learning

2.1 Introduction and terminology

In section 1.3 the RL framework was briefly discussed. In this chapter, the details of this methodology are explained.

2.1.1 Markov Decision Process

MDP is consecutive decision-making in which actions impact immediate rewards and later states, and hence future rewards. In other words, MDP is a stochastic control process using a discrete-time framework. An MDP system consist of 4 components (figure 2.1):

- *states* ($S_t \in \mathcal{S}$): A state (s) is a collection of all essential information about the current situation that can be used to forecast future states. For example, in the case of a robot arm trying to grab a box, the current position of the robot arm could be the state. States can be a multidimensional discrete or continuous set.

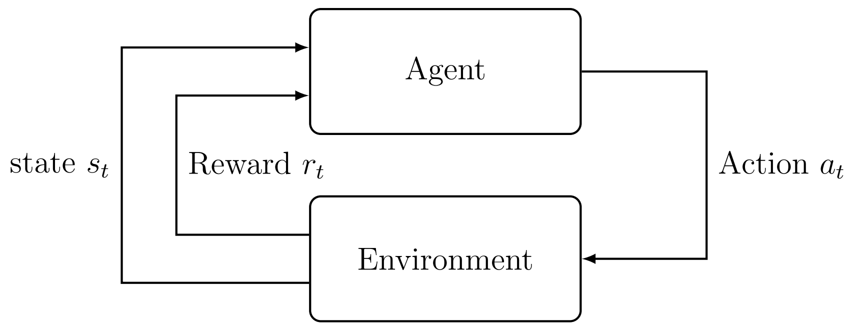

Figure 2.1: Reinforcement learning schematic and the agent environment interaction.

Sometimes, an observation of the states is available instead of the states themselves. For example, instead of a current position of the arm, a snapshot picture is available.

- *action* ($A_t \in \mathcal{A}$): Actions are utilized to control the states by the agents *policy*, which is a mapping from states to actions. It can be either stochastic $a = \pi(.|s)$ or deterministic $a = \mu(s)$. Actions somehow can be compared to the control input in the feedback of a control system. As an example, in a navigation problem, the actions are the torque applied to the wheels. Actions might belong to a discrete or continuous set, and they can also be multidimensional.

- *Reward* ($R_t \in \mathcal{R} \subset \mathbb{R}$): It is the measure of how well the agent is choosing the actions, to put it another way, how well is its *policy.* For example, in the robot arm problem, it could be how close it is to grab the box.

- *environment* (p) : The environment is fully described by its *dynamics* (distribution) which can be stochastic $s_{t+1} = p(.|s_t, a_t)$ or deterministic $s_{t+1} = p(s_t, a_t)$. Environment could be any sort of system in which a reward could be defined for a set of given actions applied to the environment.

The MDP framework is conceptual and adaptable, and it may be widely used in a variety of situations in several ways, including the stock price prediction [46] to low-level control of UAVs [47]. Therefore, the definitions are different compared to a control platform. In an MDP, the interaction between the agent (controller) and the environment (the plant, controlled unit) happens in a discrete-time steps platform. The agent performs actions (control signal), receives the reward, and ends up being in a new state. Each interaction between the environment and the agent is called a *step.* in each step the agent receives states S_t and reward $R(s_t)$ from environment and generates a set of action(s) A_t based on its policy which would transform the environment states to a new one S_{t+1} based on transition probability $P(s_{t+1}|s_t, a_t)$ and consecutively provide with a R_{t+1}. So MDP can be defined as a tuple [22]:

$$D \equiv (S, A, P, R) \tag{2.1}$$

Expected Reward can be based on the current state and action $r = r(s, a)$:

$$r(s,a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1}] \tag{2.2}$$

Or be based on the state-action-next state:

$$r(s,a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1}, S_t = s'] \tag{2.3}$$

Expected return

broadly speaking, the goal of a policy is to maximize the average reward or discounted *return* (a weighted average in which distant rewards have a less impact) in an episode*. In other words, the goal is to maximize the expected return G_t. There are different ways of defining the expected return [48], here we discuss the one with discounted rate $\gamma \in (0,1)$ in an episode with T as final time step:

$$G = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + R_T \quad 0 \leq \gamma \leq 1 \tag{2.4}$$

γ is usually a number close to one since a low γ can result in an instability [42]. If γ is chosen to be 1, then the approach is called average-reward criterion [49]. In this case, it usually cannot distinguish between short-term transient reward, and it is mostly dominated by the steady-state region. If the policy achieves both acceptable short-term and long-term optimal behavior, then it is known as bias optimal [50].

Value function

The *Value function* specifies how good a state is in an episode while a specific policy π is followed. It can be based only on the state $V^\pi(s)$:

$$V^\pi(s) = \mathbb{E}^\pi[G|s_t = s] \tag{2.5}$$

where $\mathbb{E}_\pi$ denotes the expected return given that the agent follows policy. Note that the value of expected return should be calculated until terminal state is reached. In

**episode* consists of steps, starting from initial to terminal state, when the terminal state is reached, the process starts over from the initial state.

a similar way the *State-Action Value function* of acting a in state (s) is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}^{\pi}[G|s_t = s, a_t = a] \tag{2.6}$$

The value functions are policy dependent, meaning that the value of a state could be low in a policy while it would be high in another one; having this in mind, it is obvious that the optimal value functions are the ones obeying the optimal policy π^*:

$$V^{\pi^*}(s) = \mathbb{E}^{\pi^*}[G|s_t = s] \tag{2.7}$$

$$Q^{\pi*}(s,a) = \mathbb{E}^{\pi^*}[G|s_t = s, a_t = a] \tag{2.8}$$

$Q^{\pi*}(s,a)$ is the same as having the optimal policy because given the state (s) we can obtain the optimal policy from the below equation:

$$a^*(s) = \arg\max_a Q^*(s,a) \tag{2.9}$$

Bellman equation

Bellman equation expresses the value of a state, based on the value of its successor states. Bellman equation is obeyed in all the above equations for example in the Value function we have:

$$V^{\pi}(s_t) = R(s_t, \pi(s_t)) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, \pi(s_t))V^{\pi}(s_{t+1}) \tag{2.10}$$

$$V^{\pi^*}(s_t) = R(s_t, \pi^*(s_t)) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, \pi^*(s_t))V^{\pi}(s_{t+1}) \tag{2.11}$$

In a situation with discrete actions, determining the optimal policy is simple, since an exhaustive search is possible if the optimal value function and the transition probabilities for the following states are known, however, in case of continuous spaces, function approximation methods are utilized.

There are numerous value function-based methods which has 3 major classes of:

1. Dynamic programming-based methods.

2. Monte Carlo methods
3. Temporal difference methods.

2.1.2 Dynamic programming

Dynamic Programming (DP) is well suited in a discrete scheme [51]; however, it is possible to use it in a continuous framework. DP uses value functions to arrange and guide the search for optimal policies. The transition probability of the environment should be available, or it could be determined from experience.

In a DP algorithm *policy iteration* is used, which is a process that alternates between *policy evaluation* and *policy improvement*. Initially, a random policy is used to start the approach, then the value function for the current policy is determined by policy evaluation. Each value of state in the current iteration is updated based on the values of the state in the previous iteration (bootstrapping), the policy π, and transition probability p. Finally, the policy is improved based on the most recent value function.

2.1.3 Monte Carlo methods

Unlike the DP, Monte Carlo methods learn directly from *experience*[†] with no prior knowledge of MDP transitions. They carry out rollouts by executing the existing policy on the system, which is referred to as operating on-policy. The value function is updated after an episode is ended. This process is done using the average returns using the current experiences. The frequency of transitions and rewards is recorded and utilized to calculate value function estimates. As more episodes are produced, the average value will converge. The policy is improved by making it greedy regarding the value of the states. Although the method is quite simple, it is pretty powerful; for example, in the game of Tetris, this method outperforms most of the other ones [52].

[†]sampled episodes from environment

2.1.4 Temporal Difference

Temporal Difference (TD) is a generalization of the Monte Carlo method. It also utilizes the bootstrapping of the DP so that TD(1) is the same as the Monte Carlo method, updating the values only when the episode is ended. TD(0) only considers the sampled successor states rather than the full distribution over the successor states in DP. In TD(λ) ($0 \geq \lambda \geq 1$), values are updated before the end of the episode, and more than 1 step ahead is used.

Two popular TD approaches exist, with slightly different update procedures, state-action-reward-state-action (SARSA) [53] and Q-learning [54]. SARSA uses the below equation for updating the Q value.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(x_{t+1}, u_{t+1}) - Q(s_t, u_t)]$$

While Q-learning uses:

$$Q(s_t, a_t) \leftarrow Q(s_t, u_t) + \alpha[R_{t+1} + \gamma \max_{a_{t+1}} Q(x_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.12}$$

SARSA is an on-policy algorithm, which means that its behavior and target policy are the same. Target policy is the output policy of the agent, which is used for evaluating the algorithm. The behavior policy π_b is how the agent acts in exploration. Since exploratory policies are not optimal, *on-policy* methods such as SARSA may quickly converge to a local optimum.

Off-policy agents, such as Q-learning, employ different target and behavior policies; hence, they may use equal probability for taking actions in each state, so $\pi_b(a^*|s) > 0$; as a result, they would find the optimal policy given enough time [55].

Value function methods struggle with the challenges of RL in robotic because they demand data to be filled into the entire state-action space, and they are intrinsically unstable [56]. In addition, the bootstrapping will result in a bias if we want to use function approximation techniques which is inevitable in continuous spaces of robotics. As a result, value-based methods are not suitable for robotic applications, so we introduce a new family of RL methods called the policy search in the next section.

2.2 Policy search

Policy search approaches do not require using a value function model and instead search for the optimal policy. The concept behind this method is that it is feasible to enhance an episode's return without knowing the value of each state. The disadvantage of this technique is that it requires evaluating the policy and calculating the return in order to determine if the chosen policy is superior or not. Usually, a parameterized policy is chosen, and the parameters are tuned to maximize the expected return. This is usually done by methods such as gradient ascent [57] or hill climbing [58].

Policy searches provide many advantages. For example, it is feasible to take advantage of an expert for parameter initialization [59], or it is possible to choose the suitable policy parameter structure, ensuring robustness and stability [49]. Therefore, making policy search, a well-suited method for robotic which is proven by real system applications [60, 61].

In the case where gradient ascent is used for the optimization of the policy, we have:

$$J(\pi_\kappa) = \mathbb{E}^{\pi_\kappa}[G_t] \tag{2.13}$$

$$\kappa_{k+1} = \kappa_k + \alpha(\nabla_\kappa J(\pi_\kappa)|_{\kappa=\kappa_k} \tag{2.14}$$

In which, $\nabla_\kappa G_{\pi_\kappa}|$ is called *policy gradient* [62]. Which can be expressed as:

$$\nabla_\kappa J_{\pi_\kappa} = \gamma^t G_t \frac{\nabla_\kappa \pi_\kappa(a_t|s_t)}{\pi_\kappa(a_t|s_t)} \tag{2.15}$$

$$\nabla_\kappa J_{\pi_\kappa} = \gamma^t G_t \nabla_\kappa \log \pi_\kappa(a_t|s_t) \tag{2.16}$$

The term $\nabla_\kappa \log \pi_\kappa(a_t|s_t)$ is referred to as *eligibility vector*. Equation 2.16 is first introduced by [63] known as REINFORCE algorithm. This algorithm needs the episode to be terminated to calculate G_t, which is why this algorithm is considered a Monte Carlo algorithm. Methods such as Trusted Region Policy Optimization (TRPO) [37] or Proximal Policy Optimization (PPO) [38] are examples of using such methodology.

For continuous actions, instead of learning the probability of the infinite number of actions, usually a Gaussian distribution is used:

$$\pi(a|s,\kappa) = \frac{1}{a_\sigma(s,\kappa)\sqrt{2\pi}} exp\big(- \frac{(a - a_\mu(s,\kappa))^2}{2a_\sigma(s,\kappa)^2}\big) \tag{2.17}$$

One of the methods to parameterize the policy is using a neural network named Deep Reinforcement Learning.

2.2.1 Deep Reinforcement Learning

In RL, neural networks (NN) are function approximation tools when the state or action space is continuous or too large. In some instances, it is simpler to approximate the value function, whereas, in others, it is easier to approximate policy. In latter cases, policy-based methods are more favorable as they yield a better asymptotic policy [64]. In both cases, a neural network can be employed for value approximation or policy approximation.

Neural networks can learn to map states to values or state-action pairs to Q values. Instead of using a lookup table to store, index, and update all possible states and their values - which is impossible with huge problems- We can train a neural network on samples from the state and action space to predict the value of states or which actions to take given a state.

Now that policy search is introduced, it is possible to discuss the next generation of RL, actor-critic methods.

2.3 actor-critic methods

Actor-critic methods are policy search methods in which a bias is introduced through bootstrapping in order to improve learning speed and reduce variance. The actor-critic method to the REINFORCE is like the TD algorithm to the Monte Carlo methods.

If only one step of the return is considered, (like TD(0)) the general formula for an actor-critic method can be given as:

$$\kappa_{k+1} = \kappa_k + \alpha\big(R_{t+1} + \gamma \hat{v}_\omega(S_{t+1}) - \hat{v}_\omega(S_t)\big)\nabla_\kappa \log \pi_\kappa(a_t|s_t) \tag{2.18}$$

2.4 Soft Actor Critic

Soft Actor-Critic (SAC) is an actor-critic off-policy algorithm with a stochastic policy [65, 40]. It is inspired by stochastic policy optimization and Deep Deterministic Policy Gradient (DDPG) approaches [36]. It has similarities to Twin Delayed DDPG (TD3) method [66] such that both use two clipped Q approximators. Since it is a stochastic method, it also benefits from something similar to target policy smoothing. Which makes it a potent tool in the robotic control field [67].

The main feature of the SAC algorithm is that it tries to balance a trade-off between expected return and entropy [68]. The more the entropy, the higher the exploration, and the less the entropy, the higher the expected return in the short term. This is related to the exploration-exploitation trade-off: increasing entropy leads to more exploration, speeding up learning later. It can also prevent converging to futile local optimums.

Before we can discuss the further details of the algorithm, it is necessary to discuss the details of the usage of entropy in RL.

2.4.1 Entropy-Regularized Reinforcement Learning

The entropy-regularized reinforcement learning changes the goal of RL by including an entropy term, so that the optimal policy not only aims to increase the reward but also tries to increase its entropy at each visited state [69]. The temperature parameter α balance between exploration and exploitation in such way that by increasing α the policy would try to explore more by adding a stochastic term the reward importance. The formula for this method is:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{T} \gamma^t \big(r(s_t, a_t, s_{t+1}) + \alpha H(\pi(.|s_t))\big)] \tag{2.19}$$

in which $H(\pi(.|s_t))$ is the entropy of a stochastic policy, given by:

$$H(\pi(.|s_t)) = \mathbb{E}[-\log \pi(.|s_t)] \tag{2.20}$$

So, comparing a deterministic policy to an entropy regularized policy, when multiple actions are almost equally valuable, the policy commits equal probability mass to the actions instead of choosing the most valuable action. In this framework, the state value and the state-action value should be modified:

$$V_\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t \Bigg(r\Big(s_t, a_t, s_{t+1} + \alpha H\big(\pi(.|s_t)\big)\Big)\Bigg)|s_0 = s\right] \tag{2.21}$$

$$Q_\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t \Bigg(r\Big(s_t, a_t, s_{t+1} + \alpha H\big(\pi(.|s_t)\big)\Big)\Bigg)|s_0 = s, a_0 = a\right] \tag{2.22}$$

2.4.2 SAC algorithm

The SAC algorithm is given in Algorithm 2. The Q functions are updated using the Mean Squared Bellman Error (MSBE)

$$L(\delta_i, \mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s_{t+1},d)\sim\mathcal{D}}\left[\Bigg(Q_{\delta_i}(s,a) - \underbrace{\big(r + \gamma \min Q_{\delta_{targ,j}}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\kappa(a_{t+1}|s_{t+1})\big)}_{y(r,s_{t+1},d)}\Bigg)^2\right], \tag{2.23}$$

In which D is the buffer of the algorithm in which the transitions are stored. Hence the Q functions are updated by the following gradient:

$$\delta_{i,new} = \delta_{i,old} + l_\delta \nabla_{\delta_{i,old}} \frac{1}{|B|} \sum_{(s,a,r,s_{t+1},d)\in B} \big(Q_{\delta_{i,old}}(s,a) - y(r, s_{t+1}, d)\big)^2 \quad \text{for } i = 1,2 \tag{2.24}$$

The policy is updated given:

$$\max_{\kappa} \mathop{\mathbb{E}}_{s\sim\mathcal{D},\xi\sim\mathcal{N}} \min Q_{\delta_i}(s, a_\kappa(s,\xi)) - \alpha \log \pi_\kappa(a_\kappa(s,\xi)|s), \tag{2.25}$$

The policy is updated by:

$$\kappa_{new} = \kappa_{old} + l_\kappa \nabla_\kappa \frac{1}{|\mathcal{B}|} \sum_{s\in\mathcal{B}} \Big(\min Q_{\delta_i}(s, a_\kappa(s)) - \alpha \log \pi_\kappa \left(a_\kappa(s) \middle| \, s \right) \Big) \tag{2.26}$$

sampling $a_\kappa(s)$ from Gaussian distribution of policy $\pi_\kappa(\cdot|s)$ is done by the squashed Gaussian function:

$$a_t = f_\kappa(s_t, \xi_t) \tag{2.27}$$

$$a_t = tanh(\mu_\kappa(s_t) + \sigma_\kappa(s_t) \cdot \xi_t), \qquad \xi \in \mathcal{N}(0, I) \tag{2.28}$$

However, after convergence is reached in order to evaluate the policy, the randomness term of the action is omitted to improve performance:

$$\bar{a}_t = tanh(\mu_\kappa(s_t)) \tag{2.29}$$

Algorithm 2 sac algorithm

```
 1: **Initialization**: initialize policy parameters $\theta$
 2: initialize Q-function parameters $\delta_1$, $\delta_2$
 3: initialize target network parameters $\delta_{\text{targ},1} \leftarrow \delta_1$, $\delta_{\text{targ},2} \leftarrow \delta_2$
 4: initializing the replay pool $\mathcal{D}$
 5: repeat
 6:   repeat
 7:     sample action $a \sim \pi_\theta(\cdot|s)$
 8:     observe next state $s_{t+1}$ reward $r$ and done signal $d \in [\text{TRUE}, \text{FALSE}]$
 9:     save $(s_t, a_t, r(s_t, a_t), s_{t+1}, d)$ in replay pool $\mathcal{D}$
10:     if d is TRUE then
11:       reset environment state.
12:     end if
13:   until $\mathcal{D} > \mathcal{D}_{min}$
14:   sample action $a \sim \pi_\theta(\cdot|s)$
15:   observe next state $s_{t+1}$ reward $r$ and done signal $d \in [\text{TRUE}, \text{FALSE}]$
16:   save $(s_t, a_t, r(s_t, a_t), s_{t+1}, d)$ in replay pool $\mathcal{D}$
17:   if d is TRUE then
18:     reset environment state.
19:   end if
20:   if it's time to update the parameters then
21:     for $j$ in range(however many updates) do
22:       sample a batch of transitions, $\mathcal{B} = \{(s, a, r, s_{t+1}, d)\}$ from $\mathcal{D}$
23:       Update Q-functions.
24:       Update policy.
25:       Update target networks by linearization
              $\delta_{\text{targ},i} \leftarrow \eta\delta_{\text{targ},i} + (1-\eta)\delta_i$        for $i = 1, 2$
              $\kappa_{\text{targ}} \leftarrow \eta\kappa_{\text{targ}} + (1-\eta)\kappa$
26:     end for
27:   end if
28:   evaluate the policy to check the convergence using $\bar{a}_t$
29: until convergence
30: Return $\kappa, \delta_1$ and $\delta_2$
```

Chapter 3

Simulation Environment

3.1 Introduction

Small-scale helicopters are highly nonlinear systems with complex coupling. Analyzing velocity fields around the rotor requires complicated experiments and numerical methods, which differ in each flight regime such as hover, stall, etc. [70, 71, 72]. There are numerous studies on mathematical models of the helicopter dynamics and governing equations of the forces and moments applied to it [73, 74, 75]. For this study, we have used the model already developed for the Evolution-EX helicopter in [76]. Here we briefly discuss the model development of this helicopter.

3.2 Governing equations

A combination of four subsystems describes the Evolution-EX helicopter's (EEH) dynamics: the rigid-body dynamics of the fuselage, the main rotor, the tail rotor, and the empennage. Two frameworks are defined like other dynamic problems: the body (B) and the Inertia (I) framework.

3.2.1 States and control input

The states regarding the UAV dynamics include the velocity vector $[3 \times 1]$:

$$V = [u, v, w]^T \tag{3.1}$$

and the angular velocity vector $[3 \times 1]$:

$$\omega = [p, q, r]^T \tag{3.2}$$

with respect to B and the position vector [3 × 1]:

$$p = [x, y, z]^T \tag{3.3}$$

and the Euler angles vector [3 × 1]:

$$\Theta = [\phi, \theta, \psi]^T \tag{3.4}$$

with respect to I and the input vector [4 × 1]:

$$U = [\delta_{col}, \delta_{lat}, \delta_{lon}, \delta_{ped}]^T \tag{3.5}$$

In the next section the governing equations regarding the states are discussed

3.2.2 State-space equations

The Newton-Euler equations of motion of the helicopter fuselage are defined as:

$$\dot{V} = \frac{1}{m}F - \omega \times V \tag{3.6}$$

$$\dot{\omega} = I^{-1}M - I^{-1}(\omega \times I\omega) \tag{3.7}$$

$$\dot{\Theta} = \Phi(\Theta)\omega \tag{3.8}$$

$$\dot{p} = R_b^I(\Theta)V \tag{3.9}$$

F and M are defined as vector of external forces and moments respectively. Derivation of F and M are elaborated in 3.2.4 and 3.2.5 respectively. R_b^I and Φ are linear and angular velocity transformation matrices given as follows:

$$R_b^I = \begin{bmatrix} s(\theta)c(\psi) & -c(\phi)sin(\psi) + s(\phi)s(\theta)c(\psi) & s(\phi)s(\psi) + c(\phi)s(\theta)c(\psi) \\ c(\theta)s(\psi) & c(\phi)c(\psi) + s(\phi)s(\theta)s(\psi) & -s(\phi)c(\psi) + c(\phi)s(\theta)s(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix} \tag{3.10}$$

$$\Phi = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \end{bmatrix} \tag{3.11}$$

in which s, c and t stands for "sin", "cos" and "tan" respectively. The I is the moment of inertia in which the off-diagonal terms are neglected:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{3.12}$$

In the following sections, equations regarding the derivation of forces and moments in 3.6 and 3.7 are introduced.

3.2.3 blade flapping

The dynamics of main rotor and stabilizer bar of the EEH is modeled by *hybrid model approach* [77]. In this approach $\dot{a}$ and $\dot{b}$ are tip-path-plane (TPP) longitudinal and lateral flapping angles respectively and the coefficients of first harmonic approximation in the Fourier series form. The rotor flapping state equations are:

$$\dot{a} = -q - \frac{a}{\tau_f} + \frac{1}{\tau_f}(K_u\mu_x + K_w\mu_z) + \frac{A_{lon}}{\tau_f}(\delta_{lon} + K_c c) + A_b\frac{b}{\tau_f} \tag{3.13}$$

$$\dot{b} = -p - \frac{b}{\tau_f} + \frac{1}{\tau_f}(K_v\mu_y) + \frac{B_{lat}}{\tau_f}(\delta_{lat} + K_d d) + B_a\frac{a}{\tau_f} \tag{3.14}$$

In which μ_x, μ_y and μ_z are the non-dimensional airflow components defined as:

$$\begin{aligned}\mu_x &= \frac{u - u_{wind}}{\Omega R_{mr}}\\ \mu_y &= \frac{v - v_{wind}}{\Omega R_{mr}}\\ \mu_z &= \frac{w - w_{wind}}{\Omega R_{mr}}\end{aligned} \tag{3.15}$$

And the K_u, K_v and K_w are given by:

$$K_u = 2K_\mu(\frac{4}{3}\delta_{col} - \frac{Vi}{\omega R_{mr}}) \tag{3.16}$$

$$K_v = -K_u \tag{3.17}$$

$$K_w = 16K_\mu\mu_{mr}^2\frac{sign(\mu_{mr})}{(1-\mu_{mr}^2/2) * (8sign(\mu_{mr}) + CL_\alpha\sigma)} \tag{3.18}$$

The stabilizer bar state equations c and d are TPP longitudinal and lateral flapping angles of the stabilizer bar given by:

$$\dot{c} = -q - \frac{c}{\tau_s} + \frac{C_{lon}}{\tau_s}\delta_{lon} \tag{3.19}$$

$$\dot{d} = -p - \frac{d}{\tau_s} + \frac{D_{lat}}{\tau_s}\delta_{lat} \tag{3.20}$$

3.2.4 Force derivation

The force is derived as follows:

$$F = \begin{bmatrix}F_x\\F_y\\F_z\end{bmatrix} + R_b^I\begin{bmatrix}0\\0\\mg\end{bmatrix} \tag{3.21}$$

in which:

$$F_x = -a\ T_{mr} + F_{x,fus} \tag{3.22}$$

$$F_y = b\ T_{mr} + T_{tr} + F_{y,fus} + F_{y,vt} \tag{3.23}$$

$$F_z = -T_{mr} + F_{z,fus} + F_{z,ht} \tag{3.24}$$

Main rotor thrust T_{mr} is given by:

$$T_{mr} = \mathbf{f}_{T_{mr}} + \mathbf{b}_{T_{mr}} U; \tag{3.25}$$

in which:

$$\mathbf{f}_{T_{mr}} = \frac{1}{4}\rho\pi R_{mr}^4 \Omega^2 \sigma_{mr} (C_{L_0}(\frac{2}{3} + \mu_x^2 + \mu_y^2) + C_{L_\alpha}(\mu_z - \lambda_0)) \tag{3.26}$$

λ_0 is the inflow ratio expressed as:

$$\lambda_0 = \frac{V_i}{\Omega R_{mr}} \tag{3.27}$$

σ_{mr} is the solidity factor derived by:

$$\sigma_{mr} = \frac{N c_{mr}}{\pi R_{mr}} \tag{3.28}$$

and $b_{T_{mr}}$ in 3.25 is control input coefficient term given by:

$$\mathbf{b}_{T_{mr}} = \frac{1}{4}\rho\pi R_{mr}^4 \Omega^2 \sigma_{mr} C_{L_\alpha} \begin{bmatrix} \mu_x^2 + \mu_y^2 + \frac{2}{3} & -\mu_y & \mu_x & 0 \end{bmatrix} \tag{3.29}$$

Similarly, it is possible to derive the tail rotor thrust T_{tr}

$$T_{tr} = \mathbf{f}_{T_{tr}} + \mathbf{b}_{T_{tr}} U; \tag{3.30}$$

In which $\mathbf{f}_{T_{tr}}$ is:

$$\mathbf{f}_{T_{tr}} = -\frac{1}{4}\rho\pi R_{tr}^4 n_{tr}^2 \Omega^2 \sigma_{tr} C_{L\alpha_{tr}} v_{tail} \tag{3.31}$$

and input coefficients $b_{T_{tr}}$ is given by:

$$\mathbf{b}_{T_{tr}} = -\frac{1}{4}\rho\pi R_{tr}^4 n_{tr}^2 \Omega^2 \sigma_{tr} C_{L\alpha_{tr}} \begin{bmatrix} 0 & 0 & 0 & u_{tail}^2 + w_{tail}^2 + \frac{2}{3} \end{bmatrix} \tag{3.32}$$

The normalized velocities at tail rotor can be given as:

$$\begin{aligned} u_{tail} &= \frac{u - u_{wind}}{\Omega_{tr} R_{tr}} \\ v_{tail} &= \frac{v - v_{wind} - V_{i_{tr}} + x_{fus} r}{\Omega_{tr} R_{tr}} \\ w_{tail} &= \frac{w - w_{wind} - K_\lambda V_i + x_{fus} q}{\Omega_{tr} R_{tr}} \\ \Omega_{tr} &= n_{tr} \Omega \end{aligned} \tag{3.33}$$

$F_{y,vt}$ is the vertical tail force derived by:

$$F_{y,vt} = \frac{1}{2}\rho S_{vt}\Big(C_{L_\alpha}^{vt} V_{vt}(v - v_{wind}) + v_{vt}^2\Big) \tag{3.34}$$

And horizontal tail force $F_{z,ht}$ is:

$$F_{z,ht} = \frac{1}{2}S_{ht}\Big(C_{L_\alpha}^{ht} \mid u - u_{wind} \mid w_{ht} + w_{ht}^2\Big) \tag{3.35}$$

In equation 3.34 V_vt and v_{tail} are axial and normal velocities in vertical tale defined as:

$$\begin{aligned} V_{vt} &= \sqrt{(u - u_{wind})^2 + (w - w_{wind} + x_{vt}q - K_\lambda V_i)^2} \\ v_{tail} &= v - v_{wind} + x_{vt}r - V_{i_{tr}} \end{aligned} \tag{3.36}$$

Similarly in equation 3.35:

$$w_{ht} = w - w_{wind} - x_{ht}q - K_\lambda V_i \tag{3.37}$$

$F_{z,fus}$, $F_{y,fus}$ and $F_{x,fus}$ are drag forces derived by:

$$F_{x,fus} = -\frac{1}{2}\rho S_x^{fus} V_{fus}(u - u_{wind}) \tag{3.38}$$

$$F_{y,fus} = -\frac{1}{2}\rho S_y^{fus} V_{fus}(v - v_{wind}) \tag{3.39}$$

$$F_{z,fus} = -\frac{1}{2}\rho S_z^{fus} V_{fus}(w - w_{wind} + V_i) \tag{3.40}$$

the dynamic pressure of the fuselage V_{fus} in expression 3.38 is defined as:

$$V_{fus} = \sqrt{(u - u_{wind})^2 + (v - v_{wind})^2 + (w - w_{wind} + V_i)^2} \tag{3.41}$$

3.2.5 Moment derivation

Moment includes 3 terms roll, pitch and yaw:

$$M = \begin{bmatrix} M_{roll} \\ M_{pitch} \\ M_{yaw} \end{bmatrix} \tag{3.42}$$

The effect of Q_{tr} is neglected and hence:

$$M_{roll} = (K_\beta - T_{mr} Z_{cg}) b; \tag{3.43}$$

$$M_{pitch} = (K_\beta - T_{mr} Z_{cg}) a; \tag{3.44}$$

$$M_{yaw} = Q_{mr} + T_{tr} x_{fus}; \tag{3.45}$$

Main rotor drag torque Q_{mr} is derived by:

$$Q_{mr} = \mathbf{f}_{Q_{mr}} + \mathbf{b}_{Q_{mr}} U; \tag{3.46}$$

In which:

$$\mathbf{f}_{Q_{mr}} = \frac{1}{8} \rho \pi R_{mr}^5 \Omega^2 \sigma_{mr} C_{L_\alpha} \left(\frac{C_{D0}}{C_{L_\alpha}} \left(\mu_x^2 + \mu_y^2 + 1 \right) - 2(\mu_z - \lambda_0)^2 \right) \tag{3.47}$$

$$\mathbf{b}_{Q_{mr}} = \frac{1}{8} \rho \pi R_{mr}^5 \Omega^2 \sigma_{mr} C_{L_\alpha} (\lambda_0 - \mu_z) \begin{bmatrix} \frac{4}{3} - \mu_y & \mu_x & 0 \end{bmatrix} \tag{3.48}$$

3.2.6 Induced velocity

As indicated in [78] and [79] the blade element analysis considers each blade element as a two-dimensional airfoil. The aerodynamic behavior of neighboring blade elements is independent of each other. An induced inflow velocity on each blade element should be accounted for, which is a product of the rotor wake. Analytical ways of calculating the induced velocity may be found using momentum theory, vortex theory, or nonuniform inflow calculations [78].

In general, the inflow velocity calculation is a challenging task due to its non-uniformity across the blade span; mathematical simplifications should be applied to minimize the complexity of the analysis. Finally, after determining the velocity components of the blade element, the aerodynamic forces acting on this element are calculated. The complete dynamic behavior of the blade is obtained by integrating the applied forces of the individual elements throughout the blade span. Here we use an experimental approach for induced velocity.

V_i and $V_{i_{tr}}$ are induced velocity in main rotor and tail rotor respectively. V_i is given by:

$$V_i = \frac{v_a}{\sqrt{1+\bar{\mu}^2}} \tag{3.49}$$

in which:

$$V_h = \sqrt{mg/(2\rho\pi R_{mr}^2)} \tag{3.50}$$

$$\mu = \sqrt{\mu_x^2 + \mu_y^2} \tag{3.51}$$

$$\bar{\mu} = \frac{\mu}{V_h/(\Omega R_{mr})} \tag{3.52}$$

$$V_a = -\frac{w - w_{wind}}{V_h} \tag{3.53}$$

$$v_a = \begin{cases} -\frac{1}{2}V_a - \sqrt{\frac{V_a^2}{4} - 1} & \text{if } \ V_a \leqslant -2 \\ 1 - \frac{1}{2}V_a + \frac{25}{12}V_a^2 + \frac{7}{6}V_a^3 & \text{if } \ -2 < V_a < 0 \\ -\frac{1}{2}V_a + \sqrt{\frac{V_a^2}{4} + 1} & \text{if } \ V_a \geqslant 0 \end{cases} \tag{3.54}$$

similarly, $V_{i_{tr}}$ is:

$$V_{i_{tr}} = \frac{v_{a_{tr}}}{\sqrt{1+\overline{\mu_{tr}}^2}} \tag{3.55}$$

in which:

$$V_{h_{tr}} = \sqrt{f_{F_{y,mr}}/(2\rho\pi R_{tr}^2 x_{fus})} \tag{3.56}$$

$$\mu_{tr} = \sqrt{u_{tail}^2 + w_{tail}^2} \tag{3.57}$$

$$\overline{\mu_{tr}} = \frac{\mu_{tr}}{V_{h_{tr}}/(\Omega R_{tr})} \tag{3.58}$$

$$V_{a_{tr}} = -\frac{v - v_{wind} + x_{fus} r}{V_h} \tag{3.59}$$

$$v_{a_{tr}} = \begin{cases} -\frac{1}{2}V_{a_{tr}} - \sqrt{\frac{V_{a_{tr}}^2}{4} - 1} & \text{if } \ V_{a_{tr}} \leqslant -2 \\ 1 - \frac{1}{2}V_{a_{tr}} + \frac{25}{12}V_{a_{tr}}^2 + \frac{7}{6}V_{a_{tr}}^3 & \text{if } \ -2 < V_{a_{tr}} < 0 \\ -\frac{1}{2}V_{a_{tr}} + \sqrt{\frac{V_{a_{tr}}^2}{4} + 1} & \text{if } \ V_{a_{tr}} \geqslant 0 \end{cases} \tag{3.60}$$

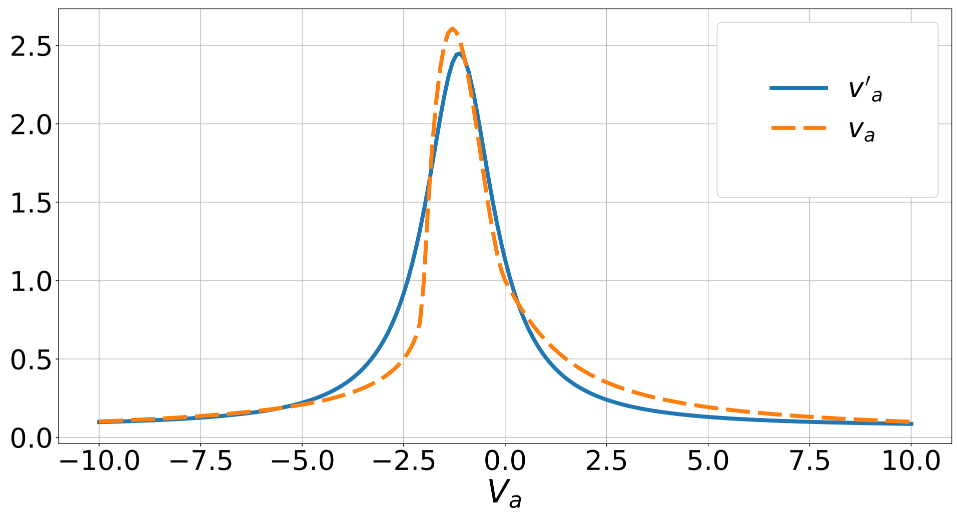

Figure 3.1: function approximation of v_a by v'_a, $R^2 = 0.947$.

Instead of using equation 3.54 and 3.60, we use the following approximate functions by regression as they provide a faster calculation time in simulations.:

$$v'_a = \frac{4.055}{(1.28V_a + 1.45)^2 + 1.7} + 0.066 \tag{3.61}$$

$$v'_{a_{tr}} = \frac{4.055}{(1.28V_{a_{tr}} + 1.45)^2 + 1.7} + 0.066 \tag{3.62}$$

figure 3.1 depicts the two functions v_a and v'_a based on V_a.

3.3 Environment setup

Now that we have discussed the dynamic of the helicopter, it is possible to set up the environment suitable for an RL process, which is developed in OpenAI Gym [80]. OpenAI gym is a software development kit for creating and comparing reinforcement learning algorithms. figure 3.2 shows the flowchart of the environment and the dashed line means that the agent is a system outside of the environment. While trying to implement an RL algorithm in a Gym environment, for each episode, first a reset function is called, then the step function is called until a terminal state is reached. In the following sections, the critical points in each part of this environment are dis-

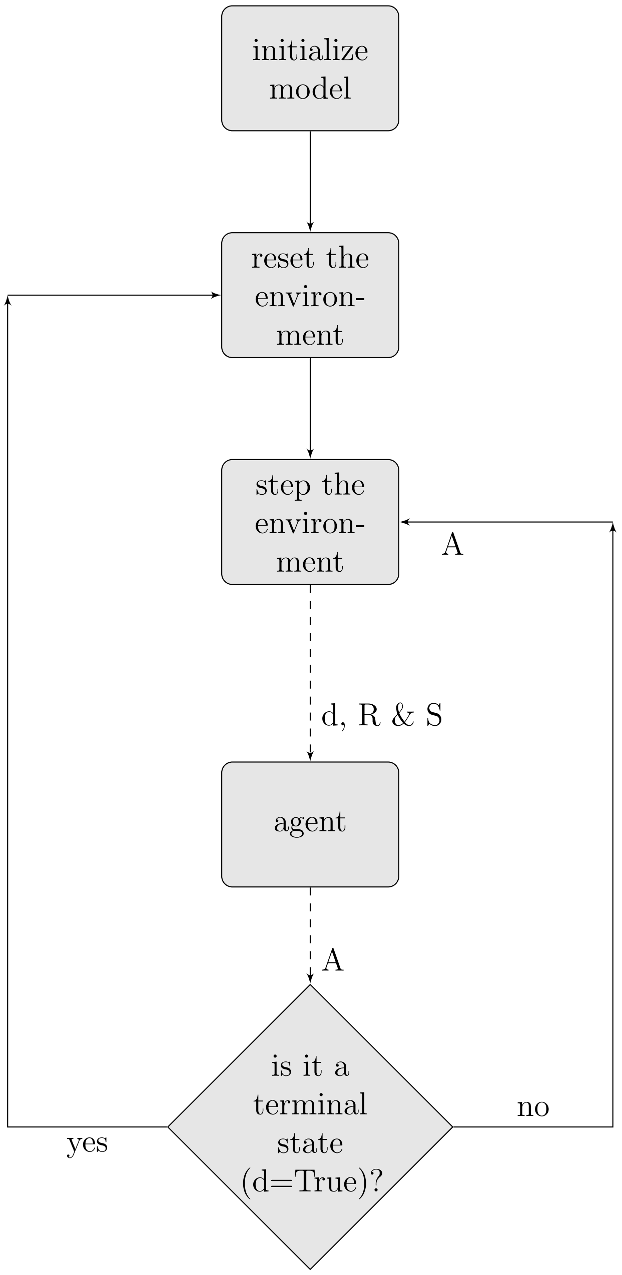

Figure 3.2: Environment Flowchart

cussed.

3.3.1 Environment Reset

Each time the environment is restarted, the helicopter is randomly placed in a position where x, y, and z are uniformly distributed in $[-1, 0, 1]$ so there would be 27 initial states. Other states are kept constant in this phase at hover state.

3.3.2 step

In each step of the episode, first, the control input is generated from the actions, then the RK45 method is used for solving the set of ODEs. In addition, the reward and the condition of reaching a final state are considered. They are elaborated in the upcoming sections.

Actions

Instead of having the 4 actions as output of the agent, 16 actions are generated by the agent in each step and the control input of the helicopter is find through the following set of equations:

$$\delta_{col} = a_1 z + a_2 w \tag{3.63}$$

$$\delta_{lat} = a_3 y + a_4 v + a_5 p + a_6 \phi \tag{3.64}$$

$$\delta_{lon} = a_7 x + a_8 v + a_9 q + a_{10} \theta \tag{3.65}$$

$$\delta_{ped} = a_{11} r + a_{12} \psi \tag{3.66}$$

This strategy would help the gradient ascent of the agent to find suitable actions for each step more easily.

3.3.3 observation

The velocity, angular velocity, location, euler angles vectors, and control input of the helicopter are all considered observations in this research. control input is considered an observation since it is not directly generated by the agent.

3.3.4 Reward

The reward function is the most important part of the environment as it provides the goal of the RL algorithm. In this research it consists of 4 terms given as follows:

$$r_t(s) = r_f + r_p + r_\psi + r_u \tag{3.67}$$

Flying term

Flying reward r_f is just a constant (18.8 in this case), assures that the algorithm is rewarded for longer episodes. The absence of this term will lead to a local minimum of reward in which the agent tries to end the episode to stop receiving negative rewards by crashing the helicopter. It also helps to stabilize the UAV in the long term.

Position term

the position error r_p punishes the agent for the distance between the current position of the UAV and the origin:

$$r_p(t) = -10\|X(t)\|_2 \tag{3.68}$$

Yaw angle term

This term also similarly punishes the agent for the error of ψ:

$$r_\psi(t) = -0.25|\psi(t)| \tag{3.69}$$

Control input terms

The control input terms consist of a derivative and a norm term to reduce chattering and increase energy consumption of the UAV:

$$r_u = -0.015\|U\| - 0.08\|U'\| \tag{3.70}$$

3.3.5 Checking for a terminal state

Unless the helicopter crashes or 8 seconds have passed, it is not a terminal state. Crashing in this research is when the states are outside of the $[-100, 100]$, except for the Euler angles which the bounds are $\phi \in [-\pi, \pi]$, $\theta \in [-\pi/2, \pi/2]$ and $\psi \in [-2\pi, 2\pi]$.

3.3.6 Summary

In this section, the dynamics for 6-DOF nonlinear dynamics of a small-scale UAV is provided. It included the effect of the fuselage, main rotor, tail rotor, etc. The setup of the environment is explained, and the code is given in Appendix A. The procedure to implement the actions and rewards in this research is also explained in detail. The implementation specifics of the SAC algorithm in this context are elaborated on in the next chapter, and the results are analyzed.

Chapter 4

Result and discussion

4.1 SAC agent

In order to solve the helicopter environment, as presented in the previous chapter, we implemented the SAC algorithm. In this section, we have provided the implementation of the soft actor-critic as a controller for the helicopter which is shown in figure 4.1. We have implemented 5 other reinforcement learning methods such as D4PG [81], proximal policy optimization (PPO) [38], Trust Region Policy Optimization (TRPO) [82], deep deterministic policy gradient [83] and Twin Delayed DDPG [66] and we were unable to find an stabilized performance of the helicopter using the aforementioned algorithms.

For this study, we use garage [84] as an API for the agent. Garage implements state-of-the-art deep reinforcement learning algorithms in Python and coherently integrates with the deep learning library PyTorch [85] and Tensorflow [86]. The library provides a straightforward approach to evaluate and test different algorithms in Gym environments. The schematic of agent-environment interaction is illustrated in Figure 4.1.

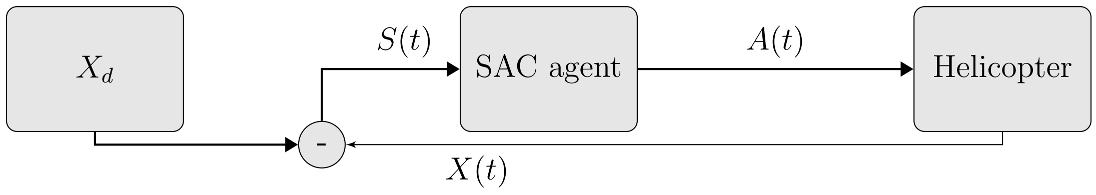

Figure 4.1: SAC controller schematic

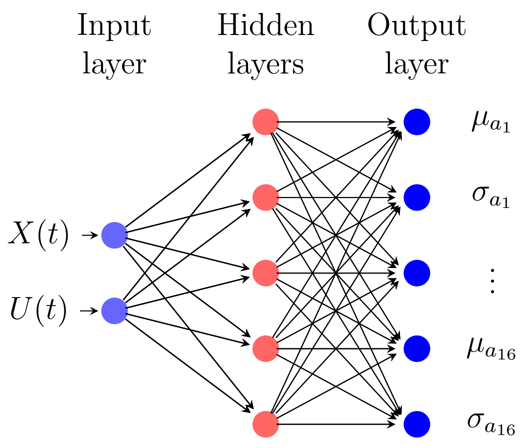

Figure 4.2: actor diagram for SAC agent, the inputs are the 16 states of the helicopter $X(t)$ and the 4 control inputs $U(t)$, the outputs are the average μ and the standard deviation σ of the agent actions

4.1.1 Architecture

In this section, we discuss the architecture of the SAC actor and critic neural network.

Actor

The actor diagram and schematic of this SAC agent is given in figures figs. 4.2 and 4.4 which includes 2 hidden layer of size 128 and 128 fully connected layers with a Rectified Linear Unity (RelU) activation function and a *tanh* activation function at last in order to narrow the result to [-1,1], the actions are then linearly mapped to the action range based on the environment[*]. In order to constraint the standard deviation of the policy, it is set to be between $[e^{-20}, e^{1}]$

Critic

The diagram for the Critic neural network is depicted in figs. 4.3 and 4.4. The diagram includes 2 hidden layer of size 256 and 256 fully connected layers with RelU activation function after each hidden layers.

4.1.2 Hyper parameters

Optuna package is utilized for optimization of all the hyperparameters of the SAC agent [87]. The SAC agent's primary hyper-parameters are given in table 4.1 as part

[*]check action wrapper at Appendix A

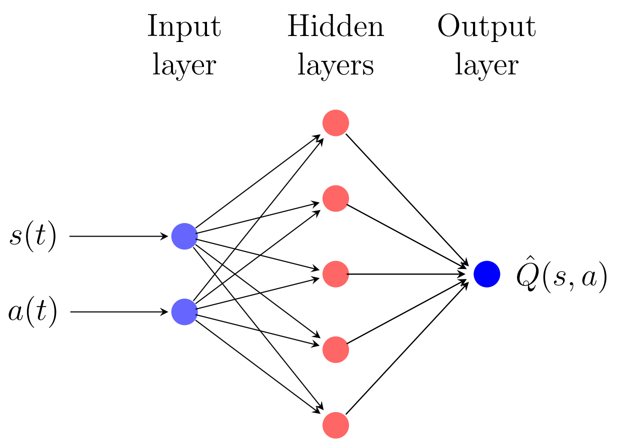

Figure 4.3: critic diagram for SAC agent, inputs are the states and actions while the output is the Q value for the given input.

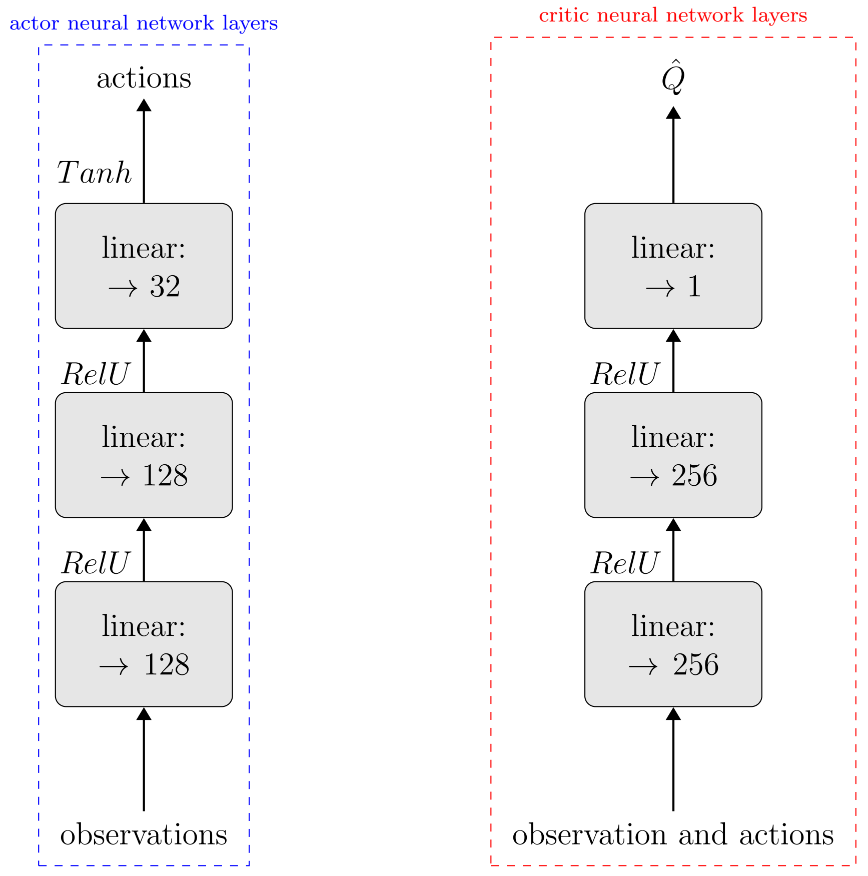
actor neural network layers
actions
$Tanh$
linear: $\rightarrow$ 32
$RelU$
linear: $\rightarrow$ 128
$RelU$
linear: $\rightarrow$ 128
observations
critic neural network layers
$\hat{Q}$
linear: $\rightarrow$ 1
$RelU$
linear: $\rightarrow$ 256
$RelU$
linear: $\rightarrow$ 256
observation and actions

Figure 4.4: actor and critic neural network diagram

of its implementation.

The replay buffer is designed to contain 10^7 transition tuples before starting the SAC algorithm. The experience replay buffer enables learning from prior policy experiences while avoiding correlated samples in the gradient step. Furthermore, we implement an upgraded target network with a target factor of 5×10^{-3}.This is motivated by the desire to improve the stability of the learning process. Based on line 21 of algorithm 2, if it is time to update, the gradient step per epoch is set to 2 and the gradient step per iteration is set to 8. For each time a gradient step is executed, a mini-batch of 2048 random samples is chosen from the replay buffer. The learning rate for Adam optimizer on both neural network is set to be 3×10^{-3}.

The training of the network included about 5×10^7 to 6×10^7 environment time steps of about 0.03, simulation time. The simulation is run in a 32-cores Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz which took about 3–4 days.

4.2 Results

4.2.1 Training and evaluation

The training result is given in 4.5. The iteration is continued for 10,000 iterations; however, no significant improvement is found after 6000. The standard deviation increases as the number of iterations increases. This is acceptable because at first, in all the episodes, the simulated UAV crashes. However, as the training progresses, the agent improves its action to stabilize the helicopter and hence the difference between the return of points closer to the origin and those placed at a more distant point from the origin grows.

4.2.2 Controllability and stability

Figure 4.6 shows the result of the initial point set to [0,0,1]. As seen, the resulting policy achieves good stabilization capability in which x,y,z stabilize after almost 2 seconds with a minor steady-state error on x and y, which is also reasonable because the helicopter cannot be stabilized at [0,0,0]. The ψ angle has a 25° overshoot,

Hyper parameter	**Value**	**description**
$\mathcal{T}$	5×10^7	Total number of environment steps.
$\mathcal{B}$	2048	mini batch which is the number of samples from the buffer randomly sampled for each stochastic gradient decent step update
$\mathcal{D}$	10^7	replay buffer size, which is the total number of steps saved in buffer (when new ones are added the oldest ones are removed.)
l_κ	3×10^{-4}	learning rate for optimization of policy by this factor.
l_δ	3×10^{-4}	learning rate for optimization of Q functions.
τ_{target}	5×10^{-3}	updating the target network linearly by this factor.
γ	0.99	factor for discounting later rewards.
α	$3 \times e^{-0.009i}$	the temperature term in SAC policy.
χ	3×10^{-3}	The learning rate of Adam optimizer.
σ_{mean}	e^{-20}	the minimum standard deviation of the actions in the stochastic policy of SAC.
σ_{mean}	e^1	the maximum standard deviation of the actions in the stochastic policy of SAC
i_{max}	10^4	the maximum number of iteration

Table 4.1: hyper parameters of SAC agent.

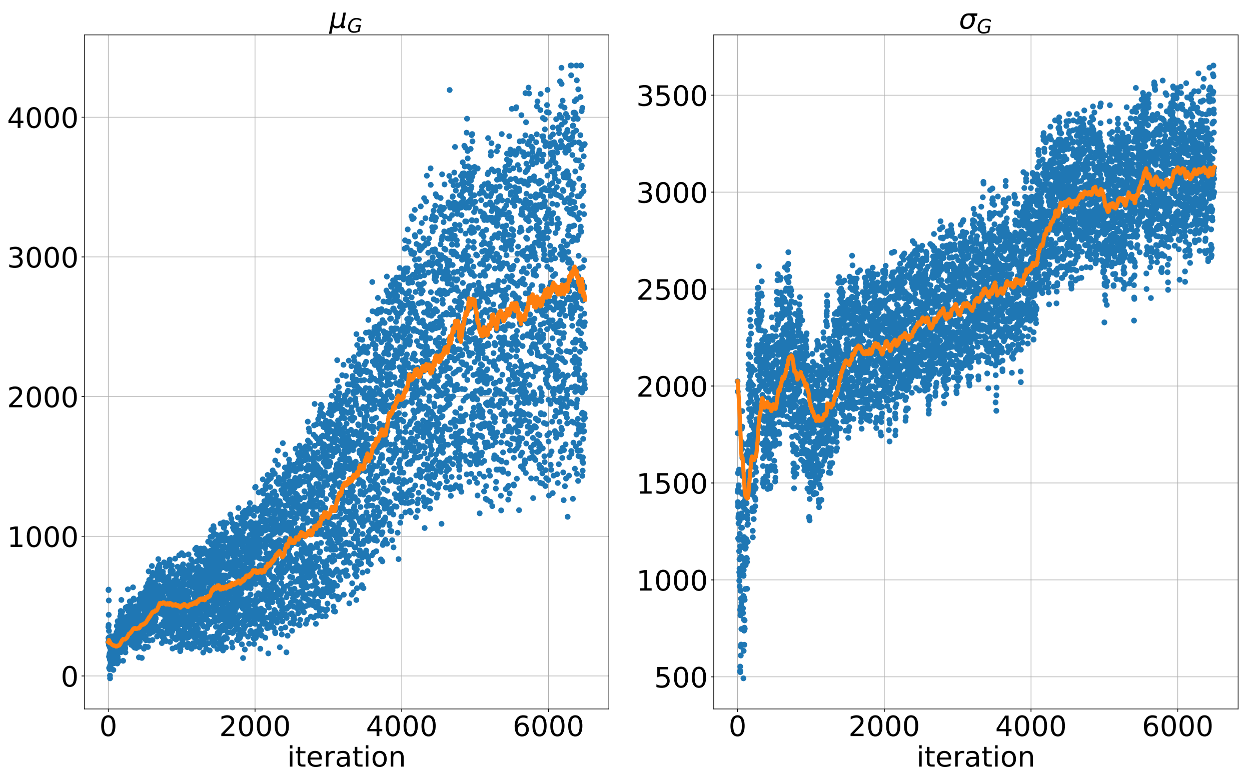

Figure 4.5: averaged discounted return μ_G and standard deviation σ_G of each iteration using the random actions of the policy.

which is also realistic because as the main rotor thrust rises, the tail rotor needs some time to adjust and minimize the ψ error, and the push of the tail rotor causes some instability to the y and finally the x states. This is a good and stable policy with a fast-rising time with minimum overshoot and steady-state error on x, y, and z states.

The result of the control input is given in figure 4.7. There are some vibrations in the δ_{ped} and δ_{col}, We find that it was somewhat hard to reduce these vibrations because as we increased the control derivative input term in the 3.70, the policy would alternate between getting closer to the target and achieving a stable hovering somewhere far from the origin.

In order to test the robustness of the policy, a wind is blown at the UAV given the following equation:

$$V_{wind,t} = V_{wind,t-1} + W \tag{4.1}$$

in which W is a random number in [-0.1,0.1] at each time step and using the policy generated it achieved 100% stability in all the 27 initial positions and the videos of them are given in git hub.

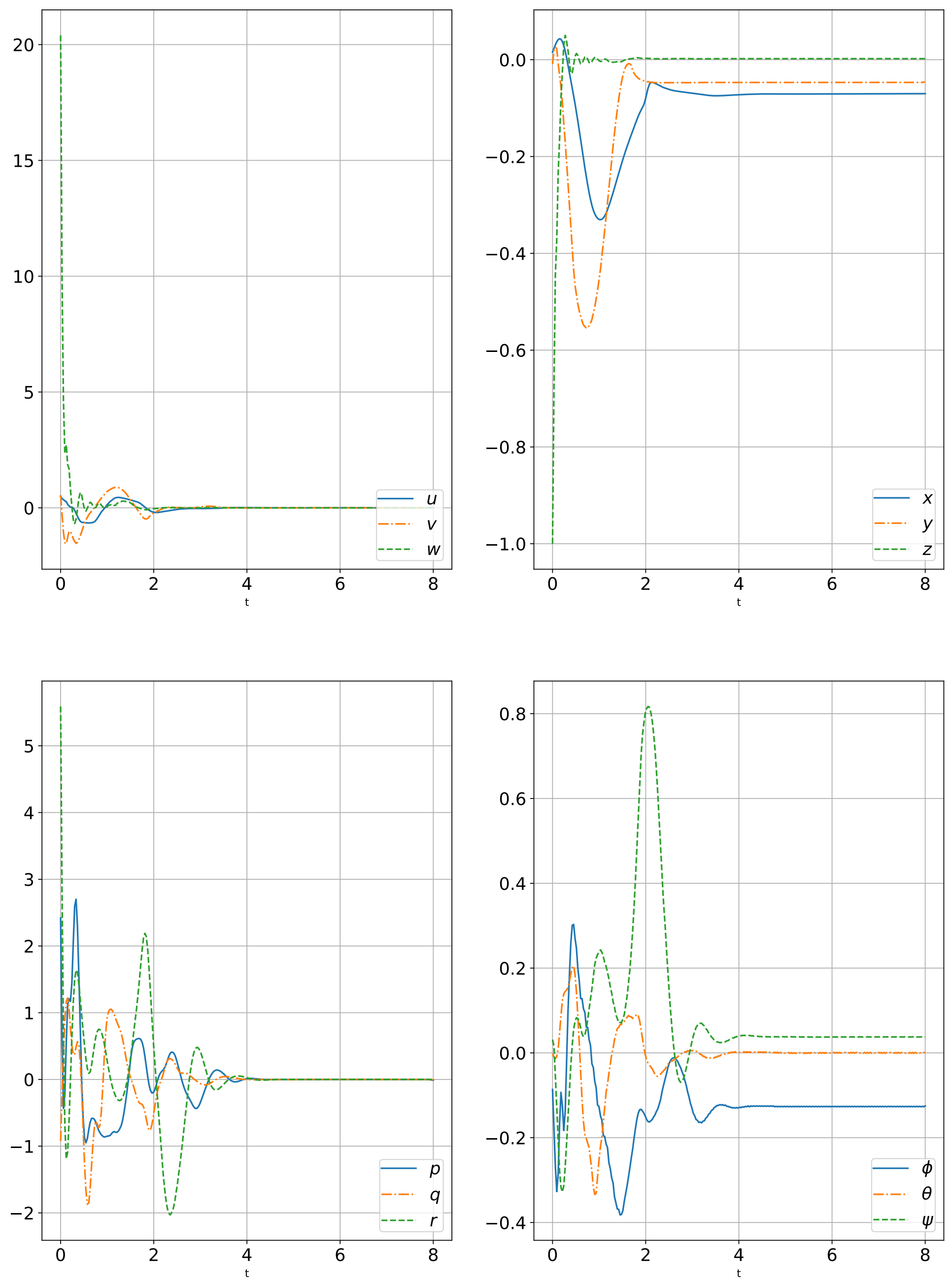

Figure 4.6: helicopter states by initial position [0,0,-1].

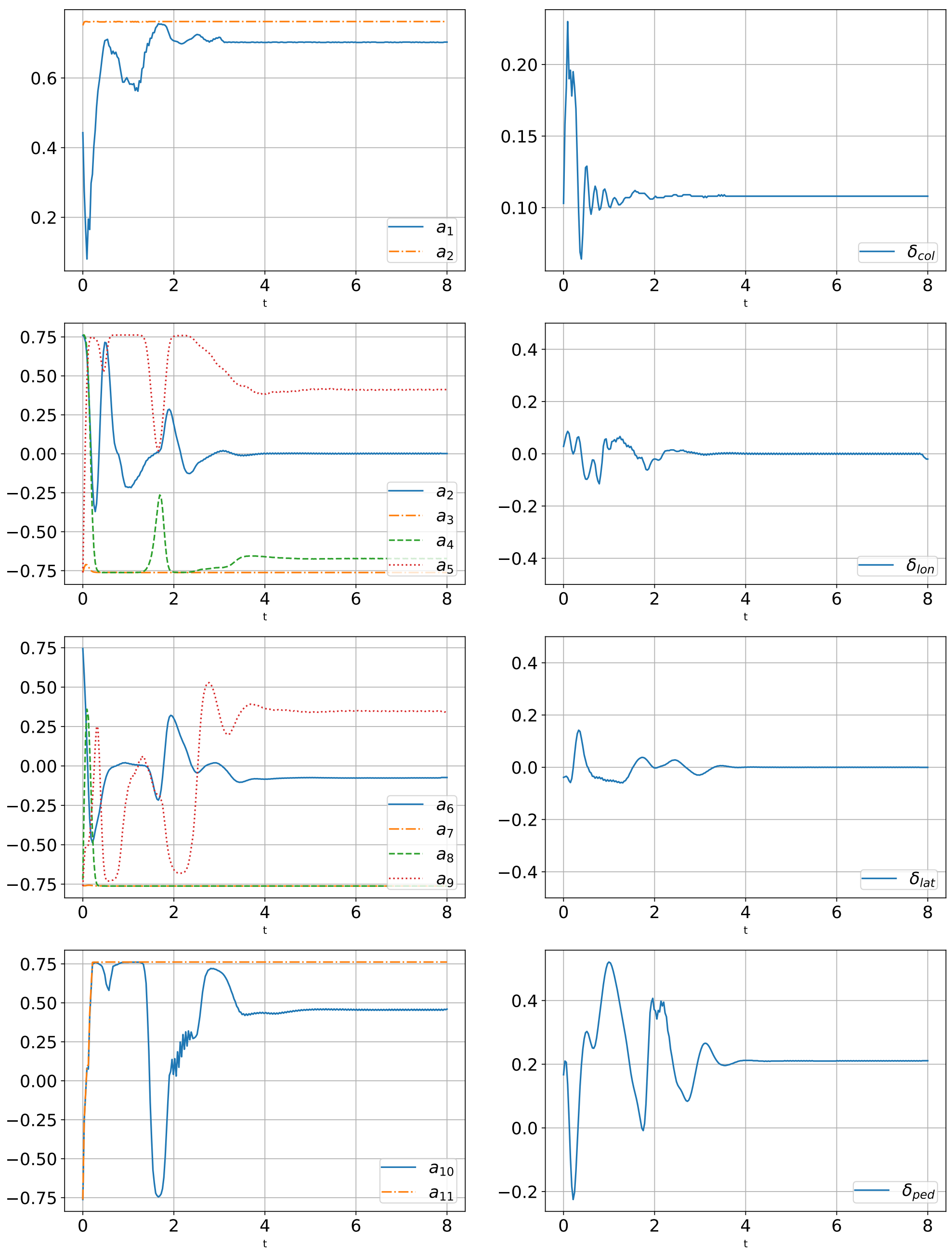

Figure 4.7: Actions and control inputs of the helicopter.

Chapter 5

Conclusions and Future Directions

5.1 Conclusions

This study shows how to train a reinforcement learning agent using a model-free off-policy technique, specifically the Soft Actor-Critic algorithm, to produce a policy capable of performing low-level control of a simulated small-scaled helicopter. The use of this method for the same task has never been disclosed previously. We also assessed the policy in an environment with the random wind as a disturbance to test the robustness of the method, and it is demonstrated that the SAC technique was capable of achieving stability in all trials.

5.2 Future work

Although it was demonstrated here that the small-sized helicopter could be stabilized using the SAC approach, trajectory tracking and recovery operations were not conducted in this study and can be addressed in future studies. In [39], similar study was conducted on a quadcopter.

The ability to efficiently apply deep RL algorithms to the real world to address practical applications may be the most compelling motivator for future advances in the area. This study showed that RL is capable of controlling the helicopter; however, this has been done in the simulation environment, future studies could be focused on using such policies in real-world data. A review of similar approaches may be found in [43].

There is a possibility of a relatively large gap between the simulation environment and the real-world data; a possible moderator would be to take advantage of a more sophisticated model such as the one for Yamaha R-50 helicopter [14, 15] to improve the replication of the environment.

Bibliography

[1] Kimon P Valavanis and George J Vachtsevanos. *Handbook of unmanned aerial vehicles*, volume 2077. Springer, 2015.

[2] Eduard Semsch, Michal Jakob, Dušan Pavlicek, and Michal Pechoucek. Autonomous uav surveillance in complex urban environments. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 82–85. IEEE, 2009.

[3] Anuj Puri. A survey of unmanned aerial vehicles (uav) for traffic surveillance. *Department of computer science and engineering, University of South Florida*, pages 1–29, 2005.

[4] Iván Maza, Fernando Caballero, Jesús Capitán, José Ramiro Martínez-de Dios, and Aníbal Ollero. Experimental results in multi-uav coordination for disaster management and civil security applications. *Journal of intelligent & robotic systems*, 61(1):563–585, 2011.

[5] Andreas Birk, Burkhard Wiggerich, Heiko Bülow, Max Pfingsthorn, and Sören Schwertfeger. Safety, security, and rescue missions with an unmanned aerial vehicle (uav). *Journal of Intelligent & Robotic Systems*, 64(1):57–76, 2011.

[6] Ebtehal Turki Alotaibi, Shahad Saleh Alqefari, and Anis Koubaa. Lsar: Multi-uav collaboration for search and rescue missions. *IEEE Access*, 7:55817–55832, 2019.

[7] Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22:22, 2011.

[8] Daniel Gheorghiţă, Ionuţ Vîntu, Letiţia Mirea, and Cătălin Brăescu. Quadcopter control system. In *2015 19th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 421–426. IEEE, 2015.

[9] Pengcheng Wang, Zhihong Man, Zhenwei Cao, Jinchuan Zheng, and Yong Zhao. Dynamics modelling and linear control of quadcopter. In *2016 International Conference on Advanced Mechatronic Systems (ICAMechS)*, pages 498–503. IEEE, 2016.

[10] Omar I Dallal Bashi, WZ Hasan, N Azis, S Shafie, and Hiroaki Wagatsuma. Unmanned aerial vehicle quadcopter: A review. *Journal of Computational and Theoretical Nanoscience*, 14(12):5663–5675, 2017.

[11] Quan Quan. *Introduction to multicopter design and control.* Springer, 2017.

[12] T John Koo and Shankar Sastry. Output tracking control design of a helicopter model based on approximate linearization. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171)*, volume 4, pages 3635–3640. IEEE, 1998.

[13] Robert Mahony, Tarek Hamel, and A Dzul. Hover control via lyapunov control for an autonomous model helicopter. In *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No. 99CH36304)*, volume 4, pages 3490–3495. IEEE, 1999.

[14] Marco La Civita, George Papageorgiou, William C Messner, and Takeo Kanade. Integrated modeling and robust control for full-envelope flight of robotic helicopters. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 1, pages 552–557. IEEE, 2003.

[15] M La Civita, George Papageorgiou, William C Messner, and Takeo Kanade. Design and flight testing of an h00 controller for a robotic helicopter. *Journal of Guidance, Control, and Dynamics*, 29(2):485–494, 2006.

[16] Sepehr Pourrezaei Khaligh. *Control-oriented modeling and system identification for nonlinear trajectory tracking control of a small-scale unmanned helicopter.* University of Alberta (Canada), 2014.

[17] Benchun Zhou, Weihong Wang, Zhenghua Liu, and Jia Wang. Vision-based navigation of uav with continuous action space using deep reinforcement learning. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 5030–5035. IEEE, 2019.

[18] S Anand and G Verweij. What's the real value of ai for your business and how can you capitalise, 2019.

[19] Manjusha Pandey. *Machine Learning: Theoretical Foundations and Practical Applications.* Springer Nature, 2021.

[20] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[21] Andrew Ng. Machine learning yearning. *URL: http://www. mlyearning. org/(96)*, 139, 2017.

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

[23] Rui Nian, Jinfeng Liu, and Biao Huang. A review on reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering*, 139:106886, 2020.

[24] Warren B Powell. Ai, or and control theory: A rosetta stone for stochastic optimization. *Princeton University*, page 12, 2012.

[25] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[27] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[28] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1, 2007.

[29] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX*, pages 363–372. Springer, 2006.

[30] Andrew Y Ng, H Jin Kim, Michael I Jordan, Shankar Sastry, and Shiv Ballianda. Autonomous helicopter flight via reinforcement learning. In *NIPS*, volume 16. Citeseer, 2003.

[31] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial intelligence*, 55(2-3):311–365, 1992.

[32] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. Acquiring robot skills via reinforcement learning. *IEEE Control Systems Magazine*, 14(1):13–24, 1994.

[33] J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE, 2001.

[34] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[35] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):1–21, 2019.

[36] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[37] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[39] Gabriel Moraes Barros and Esther Luna Colombini. Using soft actor-critic for low-level uav control. *arXiv preprint arXiv:2010.02293*, 2020.

[40] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[41] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[42] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[43] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[44] Loc D Tran, Charles D Cross, Mark A Motter, James H Neilan, Garry Qualls, Paul M Rothhaar, Anna Trujillo, and Bonnie D Allen. Reinforcement learning with autonomous small unmanned aerial vehicles in cluttered environments-" after all these years among humans, you still haven't learned to smile.". In *15th AIAA aviation technology, integration, and operations conference*, page 2899, 2015.

[45] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE international conference on computer vision*, pages 4068–4076, 2015.

[46] Jae Won Lee. Stock price prediction using reinforcement learning. In *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*, volume 1, pages 690–695. IEEE, 2001.

[47] Chen-Huan Pi, Kai-Chun Hu, Stone Cheng, and I-Chen Wu. Low-level autonomous control and tracking of quadrotor using reinforcement learning. *Control Engineering Practice*, 95:104222, 2020.

[48] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[49] Dimitri P Bertsekas. Dynamic programming and optimal control 3rd edition, volume ii. *Belmont, MA: Athena Scientific*, 2011.

[50] Mark E Lewis and Martin L Puterman. Bias optimality. In *Handbook of Markov decision processes*, pages 89–111. Springer, 2002.

[51] Lucian Buşoniu, Bart De Schutter, and Robert Babuška. Approximate dynamic programming and reinforcement learning. In *Interactive collaborative information systems*, pages 3–44. Springer, 2010.

[52] Victor Gabillon, Mohammad Ghavamzadeh, and Bruno Scherrer. Approximate dynamic programming finally performs well in the game of tetris. In *Neural Information Processing Systems (NIPS) 2013*, 2013.

[53] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

[54] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[55] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[56] Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[57] Leemon C Baird III. Reinforcement learning through gradient descent. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1999.

[58] Hajime Kimura, Masayuki Yamamura, and Shigenobu Kobayashi. Reinforcement learning by stochastic hill climbing on discounted reward. In *Machine Learning Proceedings 1995*, pages 295–303. Elsevier, 1995.

[59] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE, 2006.

[60] Marc Peter Deisenroth, Peter Englert, Jan Peters, and Dieter Fox. Multi-task policy search for robotics. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3876–3881. IEEE, 2014.

[61] Vishesh Vikas, Piyush Grover, and Barry Trimmer. Model-free control framework for multi-limb soft robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1111–1116. IEEE, 2015.

[62] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer, 1999.

[63] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[64] Ozgur Simsek, Simon Algorta, and Amit Kothiyal. Why most decisions are easy in tetris—and perhaps in other sequential decision problems, as well. In *International Conference on Machine Learning*, pages 1757–1765. PMLR, 2016.

[65] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

[66] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[67] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning, 2019.

[68] Robert M Gray. *Entropy and information theory.* Springer Science & Business Media, 2011.

[69] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[70] Piotr Doerffer and Oskar Szulc. Numerical simulation of model helicopter rotor in hover. *Task Quarterly*, 12(3):227–236, 2008.

[71] Thomas Crittenden, Dmitry Shlyubsky, and Ari Glezer. Combustion-driven jet actuators in reversed flow configurations. In *2nd AIAA Flow Control Conference*, page 2689, 2004.

[72] MT Patterson and PF Lorber. Computational and experimental studies of compressible dynamic stall. *Journal of fluids and structures*, 4(3):259–285, 1990.

[73] Gareth D Padfield. *Helicopter flight dynamics: the theory and application of flying qualities and simulation modelling.* John Wiley & Sons, 2008.

[74] Pascual Marqués and Andrea Da Ronch. *Advanced UAV Aerodynamics, Flight Stability and Control: Novel Concepts, Theory and Applications.* John Wiley & Sons, 2017.

[75] John M Seddon and Simon Newman. *Basic helicopter aerodynamics*, volume 40. John Wiley & Sons, 2011.

[76] Sepehr Pourrezaei Khaligh. Control-oriented modeling and system identification for nonlinear trajectory tracking control of a small-scale unmanned helicopter. 2014.

[77] Bernard Mettler, Mark B Tischler, and Takeo Kanade. System identification modeling of a small-scale unmanned rotorcraft for flight control design. *Journal of the American helicopter society*, 47(1):50–63, 2002.

[78] Wayne Johnson. *Helicopter theory*. Courier Corporation, 2012.

[79] Gordon J Leishman. *Principles of helicopter aerodynamics with CD extra*. Cambridge university press, 2006.

[80] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[81] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[82] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[83] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[84] The garage contributors. Garage: A toolkit for reproducible reinforcement learning research. `https://github.com/rlworkgroup/garage`, 2019.

[85] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[86] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris

Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[87] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Appendix A

The code of the helicopter environment. The full library is available at github.

Algorithm 1: Helicopter environment algorithm.

```
import sympy as sp
import numpy as np
from numpy import concatenate as concat

import gym
from gym import spaces
from env.Helicopter import Helicopter
from utils_main import save_files


class HelicopterEnv(gym.Env):
    def __init__(self):
        self.U_input = [U1, U2, U3, U4] = sp.symbols("U1:5", real=True)
        self.x_state = [
            u_velocity,
            v_velocity,
            w_velocity,
            p_angle,
            q_angle,
            r_angle,
            fi_angle,
            theta_angle,
            si_angle,
            xI,
            yI,
            zI,
            a_flapping,
            b_flapping,
            c_flapping,
            d_flapping,
            uwind,
            vwind,
            wwind,
        ] = sp.symbols("x1:20", real=True)
        self.My_helicopter = Helicopter()
        self.t = sp.symbols("t")
        self.symbolic_states_math, jacobian = self.My_helicopter.
            lambd_eq_maker(self.t, self.x_state, self.U_input)
        self.default_range = default_range = (-2, 2)
        self.velocity_range = velocity_range = (-100, 100)
        self.ang_velocity_range = ang_velocity_range = (-100, 100)
        self.ang_p_velocity_range = ang_p_velocity_range = (-100, 100)
        self.Ti, self.Ts, self.Tf = 0, 0.03, 8
        self.angle_range = angle_range = (-np.pi / 2, np.pi / 2)
        self.psi_range = psi_range = (-2 * np.pi, 2 * np.pi)
        self.observation_space_domain = {
            "u_velocity": velocity_range,
            "v_velocity": velocity_range,
            "w_velocity": velocity_range,
            "p_angle": ang_p_velocity_range,
            "q_angle": ang_velocity_range,
            "r_angle": ang_velocity_range,
            "fi_angle": angle_range,
            "theta_angle": angle_range,
            "si_angle": psi_range,
            "xI": default_range,
            "yI": default_range,
            "zI": default_range,
            "a_flapping": velocity_range,
            "b_flapping": velocity_range,
            "c_flapping": velocity_range,
            "d_flapping": velocity_range,
            "delta_col": (-10, 10),
            "delta_lat": (-10, 10),
            "delta_lon": (-10, 10),
            "delta_ped": (-10, 10),
        }
        self.states_str = list(self.observation_space_domain.keys())
        self.low_obs_space = np.array(tuple(zip(*self.observation_space_domain.values()))[0], dtype=np.float32)
        self.high_obs_space = np.array(tuple(zip(*self.observation_space_domain.values()))[1], dtype=np.float32)
        self.observation_space = spaces.Box(low=self.low_obs_space, high=self.high_obs_space, dtype=np.float32)
        self.default_act_range = (-0.3, 0.3)
        def_action = (-1, 1)
        lat_action = (-1, 1)
        self.action_space_domain = {
            "col_z": def_action,
            "col_w": def_action,
            "lon_x": def_action,
            "lon_u": def_action,
            "lon_q": def_action,
            "lon_eul_1": def_action,
            "lat_y": lat_action,
            "lat_v": lat_action,
            "lat_p": lat_action,
            "lat_eul_0": lat_action,
            "ped_r": def_action,
            "ped_eul_3": def_action,
        }
        self.low_action = np.array(tuple(zip(*self.action_space_domain.values()))[0], dtype=np.float32)
        self.high_action = np.array(tuple(zip(*self.action_space_domain.values()))[1], dtype=np.float32)
        self.low_action_space = self.low_action
        self.high_action_space = self.high_action
        self.action_space = spaces.Box(low=self.low_action_space, high=self.high_action_space, dtype=np.float32)
        self.min_reward = -13

        self.no_timesteps = int((self.Tf - self.Ti) / self.Ts)
        self.all_t = np.linspace(self.Ti, self.Tf, self.no_timesteps)
        self.counter = 0
        self.best_reward = float("-inf")
        self.longest_num_step = 0
        self.reward_check_time = 0.7 * self.Tf
        self.high_action_diff = 0.2
        obs_header = str(list(self.observation_space_domain.keys()))
            [1:-1]
        act_header = str(list(self.action_space_domain.keys()))[1:-1]
        self.header = (
            "time, "
            + act_header
            + ", "
            + obs_header[0:130]
            + ",a,"
            + "b,"
            + "c,"
            + "d,"
            + obs_header[189:240]
            + ",rew,"
            + "cont_rew,"
            + "int_rew,"
            + "si_rew,"
            + "f_rew,"
            + "dinput_rew,"
            + "input_rew,"
        )
        self.saver = save_files()
        self.reward_array = np.array((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0), dtype=np.float32)
        self.reward_limit = [
            1.00e02,
            3.40e03,
            1.34e02,
            1.51e03,
            3.28e01,
            7.78e00,
            3.15e04,
            3.09e01,
            3.00e02,
            8.46e00,
            1.52e04,
            9.27e01,
        ]
        self.constant_dict = {
            "u": 0.0,
            "v": 0.0,
            "w": 0.0,
            "p": 1.0,
            "q": 1.0,
            "r": 0.0,
            "fi": 1.0,
            "theta": 1.0,
            "si": 0.0,
            "x": 0.0,
            "y": 0.0,
            "z": 0.0,
            "a": 0.0,
            "b": 0.0,
            "c": 0.0,
            "d": 0.0,
        }
        self.save_counter = 0
        self.longest_num_step = 0
        self.best_reward = float("-inf")
        self.diverge_counter = 0
        self.numTimeStep = int(self.Tf / self.Ts + 1)
        self.ifsave = 0
        self.low_control_input = [0.01, -0.1, -0.1, 0.01]
        self.high_control_input = [0.5, 0.1, 0.1, 0.5]
        self.cont_inp_dom = {"col": (-2.1, 2, 1), "lat": (-3.2, 3.2), "
            lon": (-3.5, 3.5), "ped": (-1.1, 1.1)}
        self.cont_str = list(self.cont_inp_dom.keys())
        self.initial_states = (
            np.array(
                (
                    3.70e-04,  # 0u
                    1.15e-02,  # 1v
                    4.36e-04,  # 2w
                    -5.08e-03,  # 3p
                    2.04e-04,  # 4q
                    2.66e-05,  # 5r
                    -1.08e-01,  # 6fi
                    1.01e-04,  # 7theta
                    -1.03e-03,  # 8si
                    -4.01e-05,  # 9x
                    -5.26e-02,  # 10y
                    -2.94e-04,  # 11z
                    -4.36e-06,  # 12a
                    -9.77e-07,  # 13b
                    -5.66e-05,  # 14c
                    7.81e-04,
                ),
                dtype=np.float32,
            )
            + 0.01
        )

        self.wind1 = np.array((0, 0, 0))
        self.jk = 1

    def reset(self):
        # initialization
        self.t = 0
        self.all_obs = np.zeros((self.no_timesteps, len(self.
            high_obs_space)))
        self.all_actions = np.zeros((self.no_timesteps, len(self.
            high_action_space)))
        self.all_control = np.zeros((self.no_timesteps, 4))
        self.all_rewards = np.zeros((self.no_timesteps, 1))
        self.control_rewards = np.zeros((self.no_timesteps, 1))
        self.control_rewards1 = np.zeros((self.no_timesteps, 1))
        self.control_rewards2 = np.zeros((self.no_timesteps, 1))
        self.control_rewards3 = np.zeros((self.no_timesteps, 1))
        self.control_rewards4 = np.zeros((self.no_timesteps, 1))
        self.control_rewards5 = np.zeros((self.no_timesteps, 1))
        self.control_input = np.array((0, 0, 0, 0), dtype=np.float32)
        self.jj = 0
        self.counter = 0
        self.wind = self.wind1
        self.jk = self.jk + 0.001
        self.current_states = concat((self.initial_states, self.wind),
            axis=0)
        self.current_states[9] = self.initial_states[9]
        self.current_states[10] = self.initial_states[10]
        self.current_states[11] = self.initial_states[11]
        self.observation = self.observation_function()
        self.done = False
        self.integral_error = 0
        return np.clip(self.observation, -0.5, 0.5)

    def action_wrapper(self, current_action, obs) -> np.array:
        self.normilized_actions = current_action
        un_act = (current_action + 1) * (self.high_action - self.
            low_action) / 2 + self.low_action
        self.all_actions[self.counter] = self.normilized_actions  #
            unnormalized_action
        self.control_input[0] = un_act[0] * 5 * obs[11] + un_act[1] * 5
            * obs[2]
        self.control_input[2] = (
            un_act[2] * 5 * obs[9] + un_act[3] * 5 * obs[0] + un_act[4]
                * 5 * obs[4] + un_act[5] * obs[7]
        )
        self.control_input[1] = (
            un_act[6] * 5 * obs[10] + un_act[7] * 5 * obs[1] + un_act[8]
                * 5 * obs[3] + un_act[9] * obs[6]
        )
        self.control_input[3] = un_act[10] * 5 * obs[5] + un_act[11] * 5
             * obs[8]
        self.control_input[0] = 2.1167 * np.tanh(self.control_input[0])
           + 0.1
        self.control_input[1] = 2.03125 * np.tanh(self.control_input[1])
        self.control_input[2] = 2.02857 * np.tanh(self.control_input[2])
        self.control_input[3] = 2.2227 * np.tanh(self.control_input[3])
           + 0.18

        self.all_control[self.counter] = self.control_input

    def find_next_state(self) -> list:
        current_t = self.Ts * self.counter
        # self.wind = self.wind + 0.4 * (np.random.random() )
        self.current_states[16:19] = self.wind
        self.current_states[0:19] = self.My_helicopter.RK45(
            current_t, self.current_states[0:19], self.
                symbolic_states_math, self.Ts, self.control_input,
        )

    def observation_function(self) -> list:
        self.observation = concat((self.current_states[0:16], self.
            control_input), axis=0)
        self.all_obs[self.counter] = concat((self.current_states[0:16],
            self.control_input), axis=0)
        for iii in range(20):
            current_range = self.observation_space_domain[self.
                states_str[iii]]
            self.observation[iii] = (
                2 * (self.observation[iii] - current_range[0]) / (
                    current_range[1] - current_range[0]) - 1
            )
        return self.observation

    def reward_function(self, observation, rew_cof=[10, 0.08, 0.015]) ->
        float:
        error = -rew_cof[0] * (np.linalg.norm(observation[9:12].reshape
            (3), 4))
        if all(abs(self.current_states[9:12])) < 0.1:
            error = error + 1 - abs(observation[8])
        reward = error.copy()
        self.control_rewards[self.counter] = error

        self.control_rewards1[self.counter] = (
            0.025 * self.control_rewards[self.counter] + self.
                control_rewards1[self.counter - 1]
        )

        reward += self.control_rewards1[self.counter]
        x = self.current_states[9]
        y = self.current_states[10]
        si = self.current_states[8]
        z = self.current_states[11]
        self.control_rewards2[self.counter] = -0.1 * np.tanh(
            0.250 / ((1 + 20 * (x ** 2 + y ** 2 + z ** 2)) ** 3 / 2) *
                abs(si)
        )
        reward += self.control_rewards2[self.counter]

        self.control_rewards3[self.counter] = 5000 / self.numTimeStep
        reward += self.control_rewards3[self.counter]

        self.control_rewards4[self.counter] = -rew_cof[1] * sum(
            abs(self.control_input - self.all_control[self.counter - 1,
                :])
        )
        reward += self.control_rewards4[self.counter]

        self.control_rewards5[self.counter] = -rew_cof[2] * np.linalg.
```

```
            norm(self.control_input, 2)
        reward += self.control_rewards5[self.counter]

        self.all_rewards[self.counter] = reward

        return reward

    def check_diverge(self, reward) -> bool:
        bool_1 = any(np.isnan(self.current_states))
        bool_2 = any(np.isinf(self.current_states))
        if bool_1 or bool_2:
            self.jj = 1
            self.observation = self.all_obs[self.counter - 1]
            reward = self.min_reward - 100
            return True, reward
        if np.isnan(reward) or np.isinf(reward):
            reward = self.min_reward - 100
            return True, reward
        for i in range(12):
            if (abs(self.all_obs[self.counter, i])) > self.high_obs_space[i]:
                self.saver.diverge_save(self.observation_space_domain, i)
                self.jj = 1

        if self.jj == 1:
            return True, reward
        if self.counter >= self.no_timesteps - 1:  # number of timesteps
            return True, reward
        return False, np.clip(reward, -1000, 1000)

    def done_jobs(self) -> None:

        self.best_reward = 0
        counter = self.counter
        self.save_counter += 1
        current_total_reward = sum(self.all_rewards)
        if self.save_counter >= 1000:
            self.save_counter = 0
            self.saver.reward_step_save(self.best_reward, self.longest_num_step, current_total_reward, counter)
        if counter >= self.longest_num_step:
            self.longest_num_step = counter
        if current_total_reward >= self.best_reward and sum(self.all_rewards) != 0:
            self.best_reward = current_total_reward
            ii = self.counter + 1
            self.saver.best_reward_save(
                self.all_t[0:ii],
                self.all_actions[0:ii],
                self.all_obs[0:ii],
                self.all_rewards[0:ii],
                np.concatenate(
                    (
                        self.control_rewards[0:ii],
                        self.control_rewards1[0:ii],
                        self.control_rewards2[0:ii],
                        self.control_rewards3[0:ii],
                        self.control_rewards4[0:ii],
                        self.control_rewards5[0:ii],
                    ),
                    axis=1,
                ),
                self.header,
            )

    def step(self, current_action):
        self.control_input = current_action
        try:
            self.find_next_state()
        except OverflowError or ValueError or IndexError:
            self.jj = 1
        self.observation = self.observation_function()
        reward = self.reward_function(self.observation)
        self.done, reward = self.check_diverge(reward)
        if self.jj == 1:
            reward -= self.min_reward
        if self.done:
            self.done_jobs()
        self.counter += 1
        if np.isnan(reward) or any((np.isnan(self.observation))):
            reward = -100
            self.current_states = self.initial_states * 0 - 10
            self.observation = self.observation_function()
        return np.clip(self.observation, -100, 100), np.clip(reward,
            -1000, 1000), self.done, {}

    def make_constant(self, true_list):
        for i in range(len(true_list)):
            if i == 1:
                self.current_states[i] = self.initial_states[i]

    def close(self):
        return None
```