REINFORCEMENT LEARNING CONTROL FOR AUTOROTATION OF A SIMPLE POINT-MASS HELICOPTER MODEL

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

KADİRCAN KOPŞA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
AEROSPACE ENGINEERING

AUGUST 2018

Approval of the thesis:

REINFORCEMENT LEARNING CONTROL FOR AUTOROTATION OF A SIMPLE POINT-MASS HELICOPTER MODEL

submitted by **KADİRCAN KOPŞA** in partial fulfillment of the requirements for the degree of **Master of Science in Aerospace Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** ________________

Prof. Dr. Ozan Tekinalp
Head of Department, **Aerospace Engineering** ________________

Assist. Prof. Dr. Ali Türker Kutay
Supervisor, **Aerospace Engineering Dept., METU** ________________

Examining Committee Members:

Prof. Dr. Ozan Tekinalp
Aerospace Engineering Department, METU ________________

Assist. Prof. Dr. Ali Türker Kutay
Aerospace Engineering Department, METU ________________

Assoc. Prof. Dr. İlkay Yavrucuk
Aerospace Engineering Department, METU ________________

Assist. Prof. Dr. Kutluk Bilge Arıkan
Mechanical Engineering Department, TED University ________________

Assist. Prof. Dr. Volkan Nalbantoğlu
School of Civil Aviation, Atılım University ________________

Date: ________________

ABSTRACT

REINFORCEMENT LEARNING CONTROL FOR AUTOROTATION OF A SIMPLE POINT-MASS HELICOPTER MODEL

Kopşa, Kadircan
M.S., Department of Aerospace Engineering
Supervisor : Assist. Prof. Dr. Ali Türker Kutay

August 2018, 69 pages

This study presents an application of an actor-critic reinforcement learning method to a simple point-mass model helicopter guidance problem during autorotation. A point-mass model of an OH-58A helicopter in autorotation was built. A reinforcement learning agent was trained by a model-free asynchronous actor-critic algorithm, where training episodes were parallelized on a multi-core CPU. Objective of the training was defined as achieving near-zero horizontal and vertical kinetic energies at the instant of touchdown. During each training episode, the agent was presented a reward at each discrete time-step according to a multi-conditional reward function. Reward function was programmed to output the negative of a weighted sum of squared vertical and horizontal velocities at touchdown. The agent consists of two separate neural network function approximators, namely the actor and the critic. The critic approximates the value of a set of states. The actor generates a set of actions given a set of states, sampled from a Gaussian distribution with mean values as output set of the actor network. Updates to the parameters of both networks were calculated from accumulated gradients during each episode and applied once per episode to improve training stability. RMSProp algorithm was used for optimization. Results achieved by the agent indicates that the method is successful at guiding the point-mass helicopter to the ground with minimal kinetic energy for most initial conditions. Controls generated by the reinforcement learning agent were found to be similar to a helicopter pilot's technique.

Keywords: Reinforcement Learning, Helicopter Autorotation, Actor-Critic Algorithms, Artificial Neural Networks

ÖZ

BASİTLEŞTİRİLMİŞ BİR NOKTA-KÜTLE HELİKOPTER MODELİ OTOROTASYONU İÇİN TAKVİYELİ ÖĞRENME KONTROLÜ

Kopşa, Kadircan
Yüksek Lisans, Havacılık ve Uzay Mühendisliği Bölümü
Tez Yöneticisi : Yrd. Doç. Dr. Ali Türker Kutay

Ağustos 2018 , 69 sayfa

Bu çalışma, basitleştirilmiş bir nokta-kütle helikopterin otorotasyonda güdümü problemine bir eyleyici-eleştirici takviyeli öğrenme metodu uygulamasını sunmaktadır. Otorotasyonda bir OH-58A helikopterinin nokta-kütle matematiksel modeli oluşturulmuştur. Bir takviyeli öğrenme elemanı, modelden bağımsız bir eyleyici-eleştirici algoritma tarafından, çok çekirdekli bir işlemci üzerinde paralel şekilde eğitilmiştir. Öğrenmenin amacı, yere değme noktasında, sıfır değerine çok yakın yatay ve dikey kinetik enerjiye ulaşmak olarak tanımlanmıştır. Öğrenme evreleri esnasında her bir ayrık zaman adımında takviyeli öğrenme elemanı bir çoğul-şartlı ödül fonksiyonuna bağlı olarak ödüllendirilmekte veya cezalandırılmaktadır. Ödül fonksiyonu yere değme noktasında yatay ve dikey hızlarının karelerinin ağırlıklandırılmış toplamının negatifini vermektedir. Takviyeli öğrenme elemanı eyleyici ve eleştirici olarak iki yapay sinir ağı fonksiyon kestirimcisinden oluşmaktadır. Eleştirici, bir durum değişkeni setinin değerini tahmin etmektedir. Eyleyici çıktıları, bir Gauss dağılımının orta noktasını temsil etmektedir. Bu dağılımlardan örneklenen reel sayılar, durum değişkeni setine karşılık gelen aksiyonları ifade etmektedir. Her iki yapay sinir ağının parametrelerinin güncellenmesi, öğrenme evreleri esnasında toplanan kısmi türevlerden hesaplanmakta ve öğrenme evresi sonunda bir defa gerçekleştirilmektedir. Optimizasyon için RMSProp algoritması kullanılmıştır. Takviyeli öğrenme elemanı tarafından elde edilen sonuçlar, uygulanan metodun birçok başlangıç koşulu için nokta-kütle helikopterin otorotasyonda minimum kinetik enerji ile yere değmesini sağlamakta başarılı olduğunu göstermektedir. Takviyeli öğrenme elemanı tarafından uygulanan kontroller, bir insan pilotun helikopter otorotasyonu esnasında uyguladığı kontrollere benzerlik göstermektedir.

Anahtar Kelimeler: Takviyeli Öğrenme, Helikopter Otorotasyonu, Eyleyici-Eleştirici Algoritmalar, Yapay Sinir Ağları

To my parents, my wife, and anyone curious enough to read this thesis

ACKNOWLEDGMENTS

I would like to thank my supervisor Dr. Ali Türker Kutay for his contribution, guidance and patience throughout this thesis study. I wish to express my sincere gratitude to my committee members Dr. Ozan Tekinalp, Dr. İlkay Yavrucuk, Dr. Volkan Nalbantoğlu and Dr. Kutlu Bilge Arıkan for their helpful comments on this study.

I want to thank my friend and colleague Ilgaz Doğa Okcu for his helpful discussion on the thesis subject and details of thesis submission procedures. I would also like to thank my colleagues Özge Kapulu, Volkan Kargın, Serkan Sağıroğlu, Sinan Özcan and Mustafa Gürler for their interest in my work and their support.

I am grateful to my parents Sema Kopşa and Haluk Kopşa. Without their love and support throughout my life, none of my achievements would be possible.

Finally, I would like to thank my beautiful wife Hande Kopşa for her support, compassion, and love. I owe everything to her.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiv

LIST OF FIGURES xv

CHAPTERS

1 INTRODUCTION 1

1.1 Literature Survey 2

1.2 Motivation of the Thesis 4

1.3 Contributions of the Thesis 7

1.4 Thesis Outline 7

2 MATHEMATICAL MODEL OF AUTOROTATION DYNAMICS . . 9

2.1 Assumptions 9

2.2 Equations Of Motion 11

2.2.1 Dynamic Equations 12

2.2.2 Rotor Dynamics 12

2.2.3 Induced Velocity Calculation 14

2.2.4 Kinematic Equations 15

2.2.5 Normalization of Variables 16

2.2.6 Validation of the Model 16

3 REINFORCEMENT LEARNING FRAMEWORK 19

3.1 Reinforcement Learning Background 19

3.1.1 Markov Decision Processes 19

3.1.2 Solutions for Markov Decision Processes 22

3.2 Function Approximation 24

3.2.1 Feedforward Neural Networks 25

3.3 Backpropagation Method 29

3.4 Optimization 33

4 PROBLEM FORMULATION 35

4.1 Definition of Environment 35

4.1.1 Reward Function 36

4.2 Agent Structure 38

4.2.1 The Critic Network 39

4.2.2 The Actor Network 39

4.3 Training Algorithm 40

4.3.1 Network Parameter Updates 42

4.3.1.1 Critic Network Updates 42

4.3.1.2 Actor Network Updates 42

4.3.2 Hyperparameters of The Training Algorithm . . . 43

5 SIMULATION RESULTS 45

5.1 Interpretation Of The Results 48

5.2 Monte Carlo Simulations of Random H-V Initial Conditions . 56

5.3 Effect of Number of Training Episodes on Agent Performance 59

6 CONCLUSION 61

6.1 Future Work 64

REFERENCES 65

APPENDICES

A ASYNCHRONOUS LEARNING ALGORITHM 69

LIST OF TABLES

TABLES

Table 2.1 Point mass model parameters of OH-58A 15

Table 5.1 Autorotation Entry Conditions 47

Table 5.2 Agent performance with varying training intervals 59

LIST OF FIGURES

FIGURES

Figure 2.1 Comparison of steady state sink-rates 17

Figure 3.1 Agent - Environment Interaction in a Reinforcement Learning Setting 20

Figure 3.2 A single hidden layer feedforward neural network 26

Figure 3.3 Examples of activation functions 27

Figure 4.1 Actor-Critic Architecture 39

Figure 5.1 OH-58A H-V diagram for Gross Weight = 1360 kg, Lock number = 5.43 46

Figure 5.2 Test points on OH-58A H-V diagram 48

Figure 5.3 Autorotation trajectory for test condition 1 49

Figure 5.4 Autorotation trajectory for test condition 2 51

Figure 5.5 Autorotation trajectory for test condition 3 52

Figure 5.6 Autorotation trajectory for test condition 4 53

Figure 5.7 Autorotation trajectory for test condition 5 54

Figure 5.8 Autorotation trajectory for test condition 6 55

Figure 5.9 Monte Carlo simulation results of random autoration entry points for $m = 1360$ kg 57

Figure 5.10 Monte Carlo simulation results of random autoration entry points for $m = 1250$ kg 57

Figure 5.11 Monte Carlo simulation results of random autoration entry points for $m = 1100$ kg 58

Figure 5.12 Monte Carlo simulation results of random autoration entry points for $m = 1000$ kg 58

CHAPTER 1

INTRODUCTION

Helicopter is a unique type of aircraft. Its capabilities such as hovering, flying backwards and laterally, and executing nap-of-earth operations make it widely used for different tasks ranging from search and rescue missions to troop transportation into and out of battle zones. Even though state-of-the-art helicopter technology is more reliable and safer compared to past, emergencies do occur.

One of those emergencies for a helicopter is engine or drivetrain failure. Loss of engine power supplied to the main rotor is critical, since lift of a helicopter is generated by main rotor in contrast to an airplane where lift is supplied mostly by wings. Such an emergency can be fatal for both the crew and the helicopter if the pilot fails in taking correct actions for a safe landing.

Autorotative descent is the power-off maneuver executed by helicopter pilots for safe landing after engine failure. During autorotation, the engine is disengaged from the drive system; therefore, the rotor blades can freely rotate and the rotation of the blades is achieved by the airflow in the upward direction. Without the power supplied by the engine, this upward flow can be used to prevent main rotor rpm from slowing down to a point of total loss of lift.

As a safety measure, helicopters are certified with a Height-Velocity diagram, which designates a region to be avoided where safe landing can not be guaranteed if engine failure occurs within this zone. The regions to be avoided establishes an envelope, in which the pilot has enough total energy in terms of height and velocity that can be used to maintain rotor rpm, and arrest the sink rate for a cushioned touchdown.

Executing a successful autorotation and touchdown is arguably one of the most difficult tasks for a helicopter pilot, for it requires considerable skill, and precision in timing. The immediate effect of engine power loss on the helicopter is the slowing rotor and an angular acceleration around the yaw-axis. At this point, the pilot must reduce the collective pitch as soon as he/she is aware of the power loss. The reduced collective helps in maintaining the rotor rpm by reducing the total drag of the rotor blades. For conditions that require higher collective pitch such as high density altitude, heavy loading, hovering or climbing, this entry phase is even more critical. After the collective is lowered, the helicopter starts to descend. A steady-state sinking must be established by the pilot as the air now flows upward through the rotor disk. Decrease in the lift produced by the rotor blades due to the lowered collective is compensated by the high angle of attack due to this upward flow. This steady descent phase is achieved by the balancing of the available energy sources in the form of potential and kinetic energy of the vehicle, and the energy stored due to the rotation of the rotor blades. As the helicopter nears the ground, the pilot must execute a flare, reducing the horizontal and vertical velocities simultaneously before touchdown. The flare can be initiated by moving the cyclic stick rearward, as this action changes the orientation of the rotor disk towards the back of the helicopter and results in an increase of the total upward force, while reducing the forward speed. In other words, the kinetic energy due to the speed of the helicopter are converted into additional provided torque in the rotor, which in turn reduces sink rate. It should be noted that the added energy to the rotor due to the flare can lead to overspeeding of the rotor and should be prevented by the pilot by effectively using the collective stick. Finally, remaining energy in the rotor is used to its full extent by increased collective pitch for a minimized sink rate as the helicopter lands. This task of careful energy management in an emergency is challenging at least for most pilots.

1.1 Literature Survey

Various studies investigated the possibility of improved autorotational characteristics by proposing solutions to the problem in a wide spectrum. These proposed solutions include passive concepts such as structural modifications of rotor systems, as well as active control of helicopters in the form of specialized automatic flight control and planning systems for autorotative landing.

Rotational moment of inertia is an important factor in describing a rotor's energy storing capability. Heavier, high-inertia rotor blades are capable of storing more energy than low-inertia blades. One approach was to increase the total rotor inertia passively for better autorotational characteristics with a High Energy Rotor System (HERS) [39]. This method was applied to an OH-58A helicopter with a rotor system of modifiable inertia blades and evaluated via flight tests [11].

Required torque, or power, of a rotor to operate at a nominal angular speed is largely defined by the aerodynamics of the rotor. Without the power supplied by the engine aerodynamic drag slows down the rotor angular speed rather quickly, unless the pilot immediately reacts to the emergency by reducing the pitch angle of the rotor blades. To nullify this effect, active addition of energy to the rotor were investigated in the form of blade tip jets and flywheels [37, 23].

Solutions to the problem of safe landing after engine failure was investigated using optimization techniques by various researchers for both hover and forward flight in all-engines-inoperative and one-engine-inoperative conditions. Johnson studied autorotative descent from hover condition using numerical optimization theory [14]. An important result of his work was that the optimal trajectory after power loss in hover is purely vertical. Lee et al. extended on Johnson's work by adding inequality constraints to the numerical optimization problem and solving for optimal trajectories at both hover and forward flight [18]. It should be noted that this study used different settings for hover and forward flight conditions, as the hover condition was examined only on the vertical dimension. A comparison of calculated optimal trajectories and flight tests of the HERS system were also given in this study. Chen and Zhao investigated optimal trajectories for one-engine-inoperative condition of a twin-engine helicopter in multiple scenarios for different maneuvers including landing [6]. A similar optimal control method was applied by Aponso and Bachelder for autorotation training for pilots in a flight simulator environment [3]. They employed a real-time trajectory optimization technique to find optimal flight path and controls depending on the state of the helicopter. Bibik and Narkiewicz applied linear-quadratic control to solve the safe landing problem after engine power loss on a complex eight-degree-of-freedom dynamic model [4]. Meng and Chen also investigated the optimal control problem of autorotation for a six-degree-of-freedom rigid-body model [21]. They found that the optimal solutions for the rigid-body model has good agreement with the two-dimensional point mass model used in other references. Dalamagkidis et al. used receding horizon neural network optimization on a vertical autorotation model for model predictive control [10]. Tierney introduced the concept of a safe landing set for the flare phase of autorotation, from which a safe landing is guaranteed [32]. Yomchinda et al. built on Tierney's work by focusing on the descent phase trajectory planning in order to ensure that the helicopter enters the safe landing set before flare phase by optimally controlling the helicopter [40]. A follow-on paper by Yomchinda et al. investigated a complete path planning algorithm in real-time for autorotation in order to achieve safe landing on predefined locations [41]. Sunberg and Rogers investigated a fuzzy-logic based, model-independent, multi-stage controller for safe autorotative landing using a 6-degree-of-freedom model [25]. Abbeel et al. achieved successful autonomous autorotation of an RC helicopter by a feedback controller derived by differential dynamic programming once the task of autorotation is captured from the flight data of autorotation landings performed by an expert pilot [1]. Lee and Bang used a Q-Learning algorithm with radial basis function approximators in order to find trajectories of safe landing during autorotation of a point-mass model [19].

1.2 Motivation of the Thesis

Automatic flight control systems and flight management systems are widely used in state-of-the-art aircraft to reduce crew workload. However, under emergency conditions pilots tend to fly themselves rather than letting autopilots have control over the aircraft. While it is logical to have a human in charge for safety-critical tasks, human pilots can suffer from stress, delayed responses, and limited ability to process and use sensory input. Since a successful autorotation depends on pilot's correct and timely actions following an engine failure, it is considerably difficult for an inexperienced pilot to execute. On the other hand, autopilots have the ability to process every bit of information coming from sensors on an aircraft at all times without delay and make decisions according to this input. Therefore, autopilots that can execute autonomous autorotation have great potential in reducing crew fatalities or loss of aircraft [27].

Another potential application of autopilots which are capable of autonomous autorotation is the guiding of unmanned aerial vehicles through autorotative descent. Unmanned rotorcraft may have a hard time recovering from such an emergency during fully autonomous missions. Most experienced manned helicopter pilots follow their instincts to time their actions during an autorotation, while using available sensor data and visual cues simultaneously for a safe landing [27]. Lacking a pilot on board, it is even more challenging to land an unmanned helicopter without engine power.

In this thesis, a machine agent is trained to execute autorotation landings following total power loss. Reinforcement learning is applied to solve autorotation problem from engine failure to touchdown. Reinforcement learning defines a framework for solving problems that involve decision making, which is inherent to the problem of autorotation as it requires both planning and control. It is used to train machine agents which have the ability to act in an environment, relying on the state of the environment and itself, to affect its future state. A scalar reward signal measures whether an agent's actions are good or bad. This signal is fed back to the agent to reinforce actions that lead to positive outcomes for that particular set of states, and vice versa. In general, the aim of the reinforcement learning is to train an agent for selecting actions to maximize cumulative reward. Hence, given an objective, a machine agent can learn a control strategy required to achieve the objective directly from its observations of the problem domain, without relying on any supervision or prior knowledge [26].

Reinforcement learning – or machine learning in a broader context – has recently been used for solving many human-level tasks, including playing video games directly from pixels, exploring virtual labyrinths, and controlling physical systems. Early research on reinforcement learning focused on problems with low dimensional state and action spaces, where value of each state and action was stored separately in a table [26]. Naturally, this approach can be applied on systems with discrete states and actions only. Furthermore, if the dimensions of state and action spaces get bigger or are continuous, storing values of each state-action pair in a table becomes intractable.

Memory required for working with such a table is also inhibiting. Therefore, compact function approximators that can deal with large state and action spaces must be used to represent how good each state and/or action is and the agent's behavior function. Various studies have shown that neural networks can be successfully used as nonlinear function approximators, which can replace large tables for storing values of the states and actions [26]. This is not a coincidence, since neural networks were shown to have the ability to approximate any arbitrary function by mapping a set of inputs to a set of outputs in a black box manner [13].

Decision-making problems similar to the autorotation problem presented in this thesis appear in many fields. One of these fields involving decision making is automatic control of physical systems. In automatic control, a controller measures outputs of the system to be controlled, and calculates actions using this feedback in order to achieve desirable behavior of the system. Reinforcement learning, on the other hand, uses a scalar reward signal conceptually similar to cost functions in optimal control, in order to train an agent for optimal decision-making to control a process. Another important difference between automatic control and reinforcement learning is that a model of the system is necessary for most automatic control methods, while reinforcement learning can derive optimal decisions from observations only.

It is important to show that control of the autorotative descent from the point of engine failure to touchdown can be achieved in closed-loop by reinforcement learning while taking into account the multi-phased nature of the autorotation, i.e. planning of the maneuver. Classical control solutions to the problem would need different controllers for each phase of the autorotation and trajectory generators to follow a predetermined path. The ability of the agent trained by reinforcement learning to generalize from previous experiences also make it a viable option for the problem at hand in case of uncertainties, in contrast with most of the research in the field that focus on obtaining open-loop optimal trajectories for a predefined set of initial conditions and that have no robustness for uncertainty.

1.3 Contributions of the Thesis

This thesis studies the application of a relatively new reinforcement learning approach for the solution of continous-state continuous-action dynamical system control problem in the context of helicopter autorotation. Although similar learning algorithms have been applied to rotorcraft flight, this study differentiates from them, as the framework presented here relies only on its observations without the incorporation of expert pilot knowledge. Furthermore, this study shows that model-free control is possible for difficult tasks of aircraft such as autorotation on a simplified point-mass model. In addition, most of the previous studies on the subject focus on finding optimal trajectories to be followed by some other control architecture, while this thesis presents a feedback controller as its final product. This final form of the agent do not need interpolation of optimal trajectory solutions for each entry condition to autorotation, in contrast to aforementioned optimal solutions. Moreover, the trained agent is both a planner and a controller; thus, it does not need transition logic from one controller to another as it would be necessary for a multi-phased control architecture designed separately for each autorotation phase. Finally, the trained agent is shown to be capable of safe autorotation and landing of the point-mass helicopter model from initial conditions within the Height-Velocity curve.

1.4 Thesis Outline

Remaining chapters of the thesis can be summarized as follows: A mathematical model of the autorotation dynamics is presented in Chapter 2; reinforcement learning framework with a background of the theory and the gradient based optimization for training is explained in Chapter 3; problem formulation for autorotative descent with actor-critic reinforcement learning is described in Chapter 4; simulation results for various initial condition pairs of forward speed and altitude both inside and outside of the Height-Velocity avoid zone are given in Chapter 5. Finally, Chapter 6 concludes the thesis by summarizing the main results and discussing the future work.

CHAPTER 2

MATHEMATICAL MODEL OF AUTOROTATION DYNAMICS

A mathematical model of the underlying dynamics of helicopter autorotation was needed. For any dynamic system model there is a trade-off between model fidelity and computational efficiency. Since reinforcement learning is essentially a trial-and-error method, number of training episodes in a single run can easily be above the order of 10^6, which means marching the model forward even with large time steps on the order of 0.1 during each episode is computationally expensive. Therefore, training process for control of complex dynamic problems are rather time consuming on standard workstations. To alleviate some of this heavy computational load and reduce training time, a simple model with a fidelity high enough for capturing essential dynamics of autorotation was necessary.

In [18] a point mass model of an OH-58A light single-rotor helicopter was built. This helicopter was modified with a High Energy Rotor System (HERS), which enabled researchers to change total rotational inertia of the rotor system, and thus the rotor system Lock Number. The Bell Helicopter Company used this helicopter to perform flight tests to assess autorotation characteristics for rotors with different Lock Numbers [11]. Dynamic performance model of a helicopter in autorotation used in this study is based on Lee's work. This point mass model consists of two degree of freedom aircraft dynamics in vertical plane and rotor dynamics.

2.1 Assumptions

As the model was built to reflect physics which govern autorotation in a simplified manner, certain assumptions and simplifications were made.

Helicopter motion is constrained to vertical plane only. For faster execution of the training algorithm, state and control space dimensions were needed to be kept low. Hence, model states only in vertical and longitudinal directions were considered. Similarly, control input space consists of vertical and longitudinal components of rotor thrust, excluding lateral and directional control channels unlike conventional helicopters. To further reduce model complexity and number of states, a point mass model was used.

Induced velocity is assumed to be triangular. Dynamics of the induced velocity is neglected, so that change in inflow is independent of time. Momentum theory is used to compute induced velocity over the rotor disk at each time step. An empirical approximation is incorporated into inflow calculation at vortex ring state, where momentum theory approximation of induced velocity is no longer valid [15]. Vortex ring state is avoided during normal operation of a helicopter including autorotation; therefore, the region of approximation for vortex ring state should never be visited, and any errors introduced by this approximation can be considered negligible.

Power losses due to compressibility of air are neglected. Fuselage parasite drag is assumed to be dependent on an equivalent flat plate drag area. Rotor profile power is assumed independent of individual rotor blades' angle of attack and a constant mean profile drag coefficient is used throughout simulations. Air density is assumed constant at all altitudes during autorotation.

Ground effect is neglected. Although it is known that ground effect causes an increase in total aircraft lift when the aircraft is close to the ground, it is stated that ground effect makes only a minor difference for the autorotation performance of a helicopter [18].

Finally, the helicopter is assumed to be in steady forward flight when engine failure occurs. Both training episodes and simulations start at the instant when engine power is lost.

2.2 Equations Of Motion

To represent dynamics of helicopter autorotation in equations of motion, a coordinate system must be established. Position of the helicopter is defined by the distance from the point of engine failure projected on the ground and corresponding states for vertical and horizontal displacements are designated as h and x respectively. At the engine failure point $h = h_0$ and h is always negative until touchdown. Other model states are vertical velocity w, horizontal velocity u, and rotor angular speed Ω.

Control variables are chosen as vertical and horizontal components of the thrust coefficient C_T, and denoted as C_{T_z} and C_{T_x} respectively. This choice of controls can be treated as approximations for collective and longitudinal cyclic controls of a conventional helicopter. The angle between thrust vector and vertical axis, which also corresponds to the rotor tip path plane angle, is denoted as α. C_{T_z} and C_{T_x} can be related to C_T as:

$$\begin{aligned} C_{T_z} &= C_T \cos\alpha \\ C_{T_x} &= C_T \sin\alpha \end{aligned} \tag{2.1}$$

It is obvious that control variables C_{T_x} and C_{T_z} are interchangable with the rotor thrust coefficient C_T and rotor tip path plane angle α. Longitudinal cyclic input can be roughly approximated by the rotor tip path plane angle. Collective input at 75% span of the rotor radius can be approximated from blade element theory as given in [14]:

$$\theta_{0.75} = \frac{\left(1+\frac{3}{2}\mu^2\right)\left(6\frac{C_T}{a\sigma}\right)+\frac{3}{2}\lambda\left(1-\frac{1}{2}\mu^2\right)}{1-\mu^2+\frac{9}{4}\mu^4} \tag{2.2}$$

where μ and λ are the advance ratio and the inflow ratio at the tip path plane respectively. a is the blade lift curve slope and σ is the rotor solidity. Advance ratio μ was calculated as:

$$\mu = \frac{u\cos\alpha + w\sin\alpha}{\Omega R} \tag{2.3}$$

Here Ω is the rotor angular speed and R is the rotor radius. Calculation of inflow ratio λ is given in Equation 2.13. a, σ and R are given in Table 2.1.

2.2.1 Dynamic Equations

Force equilibrium of the point mass model in vertical and horizontal axes gives:

$$\begin{aligned} m\dot{w} &= mg - T\cos\alpha - D\sin\theta \\ m\dot{u} &= T\sin\alpha - D\cos\theta \end{aligned} \tag{2.4}$$

where m is the helicopter mass, g is the gravitational acceleration, T is the rotor thrust, D is the helicopter parasite drag, and θ is the angle between total velocity vector and the horizontal axis. $(\dot{\ })$ notation represents the time rate of change of the states. Since components of the thrust coefficient are control variables, magnitude and orientation of the thrust vector can be controlled directly. Orientation of the thrust vector (or rotor disk in space) is given by α. Magnitude of the thrust vector is given by:

$$T = C_T[\rho(\Omega R)^2(\pi R^2)] \tag{2.5}$$

Here ρ is the air density, and R is the rotor radius. The parasite drag of the helicopter is:

$$D = \frac{1}{2}\rho(u^2 + w^2)f_e \tag{2.6}$$

f_e is the equivalent flat plate drag area of the helicopter. Trigonometric functions of the angle θ in Equations 2.4 can be replaced by:

$$\begin{aligned} \sin\theta &= \frac{w}{\sqrt{u^2 + w^2}} \\ \cos\theta &= \frac{u}{\sqrt{u^2 + w^2}} \end{aligned} \tag{2.7}$$

Equations 2.4 can now be rewritten in terms of control inputs and states as:

$$\begin{aligned} m\dot{w} &= mg - C_{T_z}\rho(\Omega R)^2(\pi R^2) - \frac{1}{2}\rho w\sqrt{u^2 + w^2}f_e \\ m\dot{u} &= C_{T_x}\rho(\Omega R)^2(\pi R^2) - \frac{1}{2}\rho u\sqrt{u^2 + w^2}f_e \end{aligned} \tag{2.8}$$

2.2.2 Rotor Dynamics

Under normal operating conditions the power required is supplied by the engine. In case of an engine failure, a helicopter can use the energy stored in its main rotor for safe landing. Energy balance equation for the rotor is:

$$I_R\Omega\dot{\Omega} = P_S - P_R \tag{2.9}$$

where I_R is the total moment of inertia of the rotor around rotation axis, P_S is the power supplied to the main rotor by the engine, and P_R is the total power required by the main rotor. In the present study complete engine power loss scenario is investigated; therefore, supplied engine power P_S is assumed zero from the moment of engine power loss until touchdown. Hence, Equation 2.9 can be expressed in terms of torque as:

$$\begin{aligned} I_R\dot{\Omega} &= -Q \\ &= -[\rho(\pi R^2)(\Omega R)^2 R]C_Q \end{aligned} \tag{2.10}$$

Q is the torque required by the main rotor, and C_Q is the torque coefficient. In [18], C_Q is approximated by Equation 2.11, and a similar approximation is used in this study.

$$C_Q = \frac{1}{8}\sigma\bar{c}_d + C_T\lambda \tag{2.11}$$

where σ is the rotor solidity ratio, $\bar{c}_d$ is the mean profile drag coefficient of the rotor blades, and λ is the inflow ratio. First term in Equation 2.11 comes from an approximation to profile drag coefficient which is derived from the blade element theory [12]. NACA 0012 is assumed as the airfoil of the main rotor blades of the OH-58A helicopter; hence, mean profile drag coefficient is taken as $\bar{c}_d = 0.0087$ [12]. Second term is the result of the momentum theory, which represents the induced power required to produce thrust [17]. To incorporate the rotor stall limit, Equation 2.11 is modified such that:

$$C_Q = \left(\frac{1}{8}\sigma\bar{c}_d\right)\left[1 + \left(\frac{C_T/\sigma}{(C_T/\sigma)_{stall}}\right)^{n_s}\right] + C_T\lambda \tag{2.12}$$

When n_s is selected a large number as in [14], the profile power of the rotor increases sharply when the rotor loading (C_T/σ) is greater than the stall limit. A value of $n_s = 20$ is used here. Stall limit for the rotor loading is 0.15 for the OH-58A helicopter as stated in [18].

Inflow ratio λ is defined as:

$$\lambda = \frac{u\sin\alpha - w\cos\alpha + v}{\Omega R} \tag{2.13}$$

where v is the induced velocity of the rotor disk. Positive direction of induced velocity is downward across the rotor disk and direction of the induced velocity vector is always opposite of the direction of the thrust vector.

2.2.3 Induced Velocity Calculation

Rotor induced velocity is given in [14] as a differential equation as follows:

$$\tau \dot{v} + v = K_{ind} v_h f_I f_G \tag{2.14}$$

where τ is a time constant of the order 0.14 for OH-58A helicopter used here [17], and is therefore neglected, K_{ind} is an empirical factor which is given by approximately 1.13 for triangular downwash distribution [18], v_h is the reference induced velocity at hover, f_I is the ratio of the induced velocity at a given flight condition to v_h, and f_G is the ground effect factor which is taken as unity, as it is ignored for this study.

Definition of the hover induced velocity v_h is:

$$v_h = \sqrt{\frac{T}{2\rho \pi R^2}} \tag{2.15}$$

It is safe to assume for the present study that thrust generated by main rotor at hover is equal to the total weight of the helicopter, i.e. $T = mg$ in Equation 2.15.

Parameter f_I is calculated as given in [14]:

$$f_I = \begin{cases} 1/\sqrt{b^2 + (a + f_I)^2} & \text{if } (2a+3)^2 + b^2 > 1 \\ a(0.373a^2 + 0.598b^2 - 1.991) & \text{otherwise} \end{cases} \tag{2.16}$$

where parameters a and b are:

$$a = \frac{u \sin\alpha - w \cos\alpha}{v_h} \tag{2.17}$$

$$b = \frac{u \cos\alpha + w \sin\alpha}{v_h} \tag{2.18}$$

The first expression in Equation 2.16 is a result of the momentum theory [12]. Outside of the region defined by the first expression is the vortex ring state, which is approximated empirically by the second expression [14]. Equation 2.16 is solved iteratively at each time step.

2.2.4 Kinematic Equations

Kinematic equations for the point mass model are:

$$\dot{h} = w \tag{2.19}$$

$$\dot{x} = u \tag{2.20}$$

Since the objective of the control method presented here is to touch the ground ($h = 0$) with minimized vertical and horizontal speeds, state h is an input to the controller. Since the controller needs to know when to flare or whether touchdown is imminent depending on h, Equation 2.19 should be included in the dynamic model. On the other hand, there is no constraint on the state x, since there are no predefined spots for the helicopter to land on. Hence, the distance traveled horizontally is trivial for the presented problem and Equation 2.20 can be omitted.

Values of the parameters used in the point mass model of OH-58A helicopter are given in Table 2.1.

Table 2.1: Point mass model parameters of OH-58A

Parameter	Value
ρ, air density, kg/m^3	1.225
R, rotor radius, m	5.37
f_e, equivalent flat plate area, m^2	2.23
σ, rotor solidity	0.048
c_d, mean profile drag coefficient	0.0087
m, helicopter mass, kg	1361
I_R, rotational inertia of the rotor, $kg\cdot m^2$	436.71
Ω_0, nominal rotor angular speed, rad/s	37.07
a, rotor blade 2-D lift curve slope	5.73

2.2.5 Normalization of Variables

As in numerical optimization techniques, stability of gradient based learning algorithms also depend on the appropriate scaling of the variables. Therefore, the states involved in the equations of motion for the problem were scaled by using the rotor nominal angular speed Ω_0 and the rotor radius R. Various scaling factors were also used to bring the nondimensional quantities to the order of 1. Normalized and scaled state variables are given in Equations 2.21.

$$\begin{aligned}
x_1 &= \frac{w}{0.1\Omega_0 R} \\
x_2 &= \frac{u}{0.1\Omega_0 R} \\
x_3 &= \frac{\Omega}{\Omega_0} \\
x_4 &= \frac{h}{10R} \\
x_5 &= \frac{C_x}{0.01} \\
x_6 &= \frac{C_z}{0.01}
\end{aligned} \tag{2.21}$$

2.2.6 Validation of the Model

In order to validate the point-mass model detailed in previous sections, steady state autorotation sink rates obtained from flight tests of OH-58A helicopter were compared with those computed by the point-mass model. Figure 2.1 shows that the point-mass model sink rates resemble those obtained from the flight tests. Flight test results for steady state autorotative descent were taken from [11].

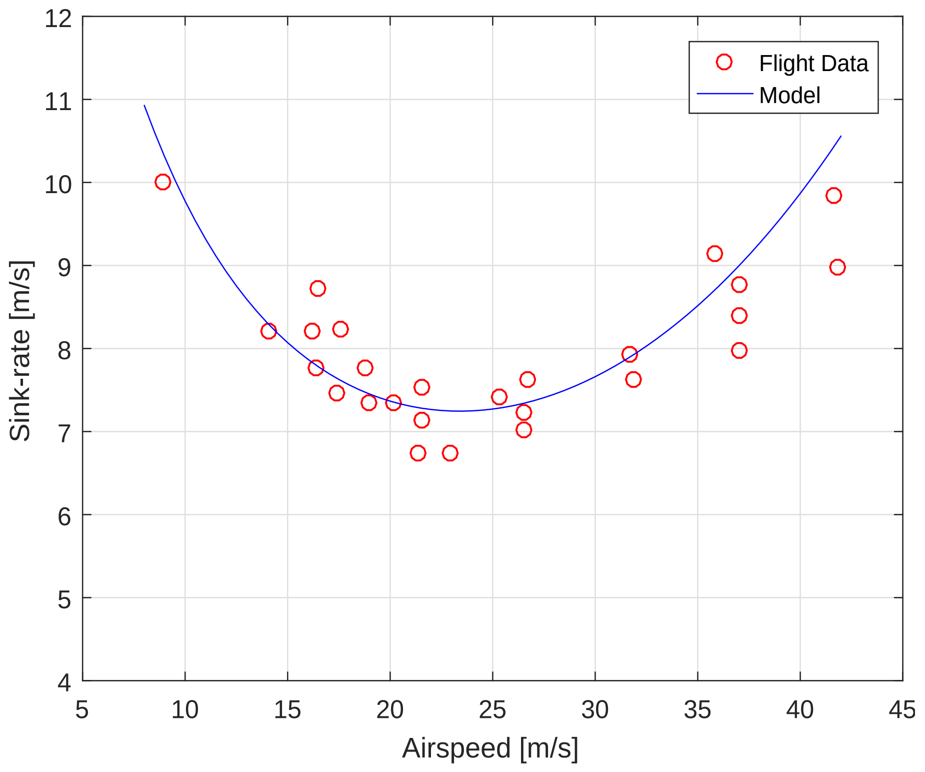


Figure 2.1: Comparison of steady state sink-rates

CHAPTER 3

REINFORCEMENT LEARNING FRAMEWORK

Traditional reinforcement learning setting is based on the assumption that both state and action spaces are finite and small. However, many real world problems are in continuous domain, such as the autorotation problem presented here. The reinforcement learning framework built for this thesis addresses the training of an agent to deal with the real-valued, continuous state and action spaces that is inherent to physical system control.

3.1 Reinforcement Learning Background

Reinforcement learning schema can be described with two main components, namely the agent and the environment. The agent is the entity that learns to act on the environment as desired by the designer. The environment is the system on which the agent acts upon. A depiction of the agent-environment interaction can be seen in Figure 3.1 In the context of this thesis, terms such as agent, environment and action were used, which corresponds to the control system engineering terms as controller, plant, and control signal respectively, in order to stay consistent with the broader field of reinforcement learning.

3.1.1 Markov Decision Processes

A state is defined as Markov, if it has all the relevant information coming from the history of the states that precedes it [26]. In other words, a Markov state represents information of all of the states that had been visited previously before reaching to the

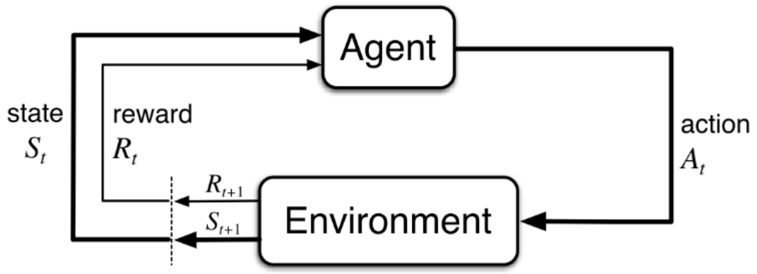


Figure 3.1: Agent - Environment Interaction in a Reinforcement Learning Setting

current state. For example, a mid-game placement of chess pieces on the board is a Markov state, as it contains all of the information from past moves. A more formal definition of the Markov property can be made as: A state S_t is called Markov if and only if $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, \ldots, S_t]$.

A reinforcement learning problem is called a Markov decision process (MDP) if each state of the environment has the Markov property. In the context of continuous-space continuous-action reinforcement learning, an MDP can be defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. Here $\mathcal{S} \in \mathbb{R}^n$ is the state space and $\mathcal{A} \in \mathbb{R}^m$ is the action space. $\mathcal{T}$ is a state transition function which is defined as $\mathcal{T}(s, a, s') = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$, where t denotes a discrete time step, s is a state sampled from the state space $\mathcal{S}$, a is an action sampled from the action space $\mathcal{A}$, and s' describes a new state. As implied by the definition, state transition function $\mathcal{T}$ is equal to the probability of transition from a state s to another state s', given an action a. For the continuous state space $\mathcal{S}$, the transition function $\mathcal{T}$ becomes a probability density function. $\mathcal{R}$ is a deterministic reward function which gives the expected reward for transitioning from one state to another under an action, i.e. $\mathcal{R}(s, a) = \mathbb{E}[r_{t+1}|S_t = s, A_t = a]$. Finally, $\gamma \in [0, 1]$ is a discount factor. Return of an MDP at any state s is defined as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3.1}$$

As it is shown in Equation 3.1, discount factor γ is a measure of scaling future rewards compared to the most recent reward. With values of γ closer to 1 delayed rewards contribute more to the return R_t, while small values of γ prioritizes more immediate rewards.

If the transition function $\mathcal{T}$ is deterministic, i.e. taking an action a in a state s always leads the process to the same next state s', the MDP is considered deterministic. From a system dynamics point of view, a deterministic transition function is roughly equivalent to the dynamic equations describing a system in the form of $s' = f(s,a)$. For time-invariant systems, an MDP describing a reinforcement learning task is called a stationary MDP. An MDP is described as fully observable if all of the states and actions relevant to an MDP are observable by the agent. This property holds generally for mathematical models of dynamic systems to be controlled, as such systems are modeled in sufficient detail by including all of the relevant information for the problem solution.

The decision-making (or action-selection) of an MDP is described by a policy π. A policy for an MDP represents the behavior of an agent and represents a mapping from states to actions of an MDP. For a continuous action space, a stochastic policy function is defined as a probability density function over the action space. Stochastic policies are necessary for any reinforcement learning task, as the agent needs to explore action space. One exploration method for continuous action spaces is Gaussian exploration, where actions are sampled from a normal distribution around an approximated mean value with a predefined or adaptive standard deviation.

Value functions of an MDP describe how good or bad each state and/or action is, in terms of future reward. There are two types of value functions: the state-value function and the action-value function. The state-value function gives the expected return starting from a state s following policy π, whereas the action-value function is the expected return starting from state s, and taking action a under a policy π. The state-value function $V^{\pi}(s)$ and action-value $Q^{\pi}(s,a)$ are:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_t | S_t = s] \tag{3.2}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_t | S_t = s, A_t = a] \tag{3.3}$$

Equations 3.2 and 3.3 can be expressed as Bellman equations using the definition of return in Equation 3.1. For a given policy π, Bellman equations for state-value and action-value functions can be written as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s] \tag{3.4}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[r_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{3.5}$$

Recursive nature of the Bellman equations of value functions gives insight for the solution of MDPs. Iterative methods can be used to exploit the recursiveness in a backward manner at each time step. An MDP is considered solved when the optimal value function is found. Value functions that yield maximum return over all policies of an MDP are called optimal value functions and defined as follows:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \tag{3.6}$$

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{3.7}$$

Another way of solving MDPs is the approximation of an optimal policy π^{*}. Policy improvement theorem states that there is always a deterministic optimal policy π^{*}, which is equal to or better than all other policies [26]. According to the theorem, the optimal policy achieves the optimal value functions $V^{*}(s)$ and $Q^{*}(s,a)$.

3.1.2 Solutions for Markov Decision Processes

Value-based model-free methods are used to approximate optimal value functions $V^{*}(s)$ and $Q^{*}(s,a)$ directly, from which an optimal policy can be indirectly obtained. These methods represent value functions using function approximators. One reinforcement learning method to approximate the optimal action-value function for control is Q-learning [34]. In Q-learning an action-value function with parameters θ^{q} is defined as $Q(s,a;\theta^{q})$. The goal of the Q-learning algorithm is to update the parameters of the function approximator by minimizing a loss function at each step until convergence to the optimal action-value function $Q^{*}(s,a)$. For the one-step Q-learning method, this loss function is defined as follows:

$$L = \left(r + \gamma \max_{a'} Q(s', a'; \theta^{q}) - Q(s, a, \theta^{q}) \right)^{2} \tag{3.8}$$

This method employs a regression on the target which is defined as the first two terms of the loss function in Equation 3.8, while the prediction of the function approximator is the last term. Exploration for this method is generally achieved by the ε-greedy method. One drawback of this method is that it has a practical application only on finite, discrete action spaces. Since action selection via ε-greedy method relies on evaluation of action-values of all possible actions at each state and choosing the action with the maximum action-value for exploitation, it is less suitable for continuous action-space problems.

Another class of methods for solving MDPs is policy-based model-free methods, where the policy is approximated directly. This time, policy function $\pi(s;\theta^q)$ is parameterized and parameters are updated by a gradient ascent on an expected value of the return. REINFORCE algorithm is an example of the policy-based methods, where the parameters are updated in the direction of policy gradient $\nabla_{\theta^q} \log \pi(s;\theta^q) R_t$ [38]. This gradient term is an unbiased estimate of the gradient of the return R_t with respect to the parameters θ^q [22]. In [38], it is shown that variance of this estimation can be reduced by subtracting a function of the state known as the baseline $b_t(s)$ from the return. Then, the total gradient for parameter update becomes $\nabla_{\theta^q} \log \pi(s;\theta^q)(R_t - b_t(s))$.

Value function estimates can be used as the baseline to reduce the variance, i.e. $b_t(s) \approx V^{\pi}(s)$ [22]. This leads to a scaling of the gradient of the policy by a term that can be expressed as the so called advantage. The advantage is defined as $A(s,a) = Q(s,a) - V(s)$, which can be interpreted as the difference in value between different actions at the same state. When return R_t is used as an estimate of $Q(s,a)$, the gradient for policy parameter updates can be redefined as $\nabla_{\theta^q} \log \pi(s;\theta^q) A(s,a)$.

3.2 Function Approximation

Early research on reinforcement learning employed lookup tables for representation of values for different state-action pairs. However, for an MDP that has large or continuous state and action spaces, using such lookup tables are not practical. Most state-of-the-art reinforcement learning researchers use function approximators for storing policy and value functions in a compact manner.

One class of function approximators are linear approximators. Linear function approximators can be written in a general form such as

$$f(s) = \theta^T \phi(s) \tag{3.9}$$

where θ is a vector of tunable parameters and $\phi(s)$ is a feature vector at state s. Feature vectors are a function of a given state which maps states into predefined features in a feature space Φ. A discussion of feature selection can be found in [5]. Linear function approximators provide insight into convergence properties of reinforcement learning algorithms, since they are easier to understand and linear in parameters. Furthermore, linear approximators are simple to implement and cheaper than nonlinear approximators in terms of computational resources. In reinforcement learning problems with large or continuous state spaces, value functions of each state can be represented with a linear approximator where the features can be taken as state variables. However, this approach assumes that the value function for a given problem is a linear combination of the state variables, which limits the solution to the set of value functions in the form given in Equation 3.9. For problems that a useful feature vector is hard to obtain, utility of linear approximators are limited.

The fact that linear approximators need hand-crafted features is a major disadvantage. For model-free reinforcement learning methods the aim is to find solutions to an MDP without relying on information about the problem domain. On the other hand, nonlinear function approximators have been used for solving reinforcement learning problems in a model-free fashion. While the same convergence properties can not be easily obtained for nonlinear approximators, it is shown that good results can be achieved for approximation of unknown functions inherent to the reinforcement learning. One example of nonlinear approximators is the artificial neural network, which had been used for many reinforcement learning tasks successfully, such as backgammon [28, 29, 30], robotics [2, 20, 33, 7], and elevator dispatching [8, 9]. In this thesis, the function approximation method was also chosen as a neural network and a discussion of neural networks is given next.

Neural networks are inspired by biological nervous systems. As in these systems, they are composed of simple elements working in parallel, with the aim of recognition of more complex concepts. Neural networks have a layered structure where each layer consists of units (or neurons) which hold simple activation functions. Between each layer there are sets of weights which connects one layer to the next. Weights of a neural network are the parameters for training or tuning a neural network for its desired purpose. The objective of training a neural network is therefore to find suitable weights for the problem at hand. Neural networks have been successfully used for several tasks, including image and speech recognition, regression, classification, and function approximation. There are three main subclasses of neural networks, namely, feedforward neural networks, convolutional neural networks, and recurrent neural networks. This study focuses on function approximation with feedforward neural networks.

3.2.1 Feedforward Neural Networks

Feedforward neural networks consist of an input layer, one or more hidden layers, and an output layer. An example of a single hidden layer feedforward neural network is depicted in Figure 3.2. This configuration of a neural network will be used as a running example for an explanation of how feedforward neural networks work in general. The output of a single hidden layer network can be written in functional form as:

$$y_i = g\left(\sum_{l=1}^{L} w_{il} f\left(\sum_{j=1}^{n} v_{lj} x_j + v_{l0}\right) + w_{i0}\right) \tag{3.10}$$

where $i = 1, 2, \dots, m$. Here, x_j is the jth input variable, v_{lj} is the weight connecting input x_j to hidden layer neuron h_l, w_{il} is the weight connecting hidden layer unit h_l to output layer neuron o_i, and v_{l0} and w_{i0} are biases for hidden and output layers respectively. $f(.)$ and $g(.)$ are activation functions that can be selected differently for the specific purpose of the neural network.

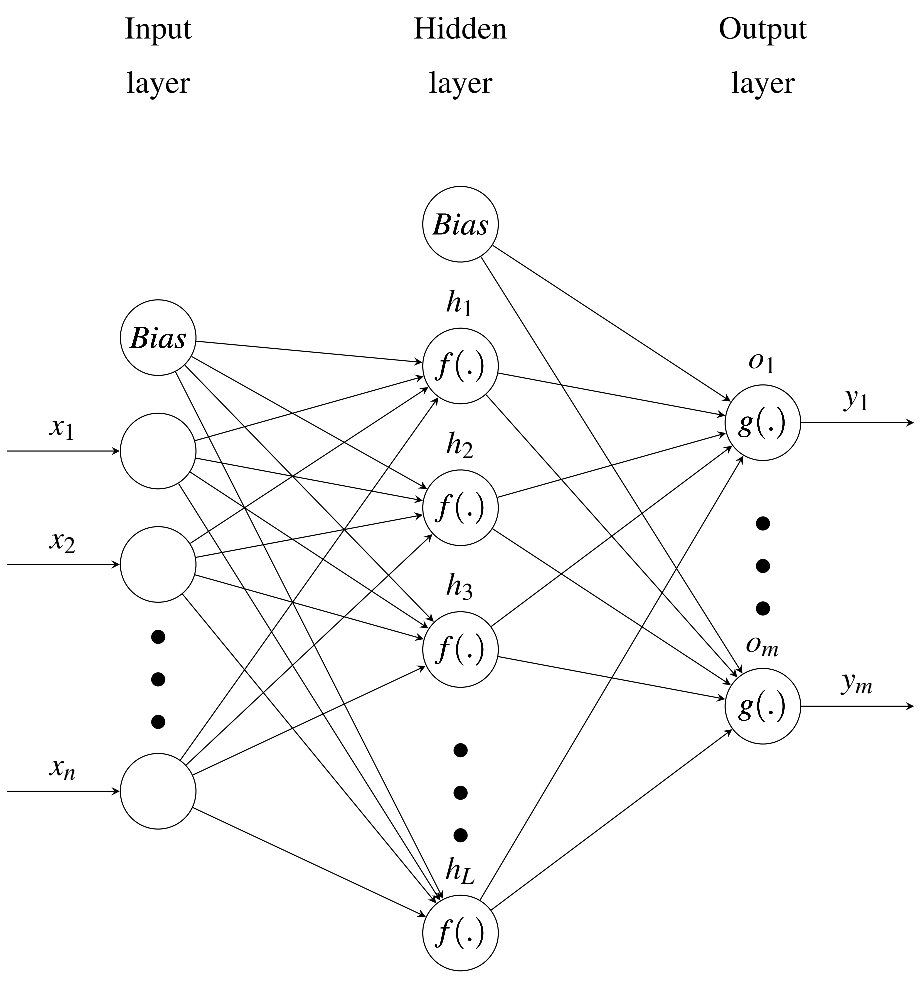
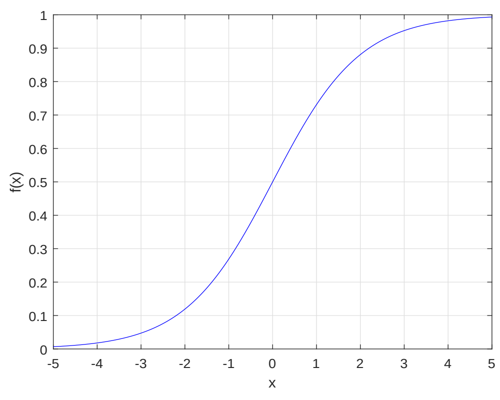


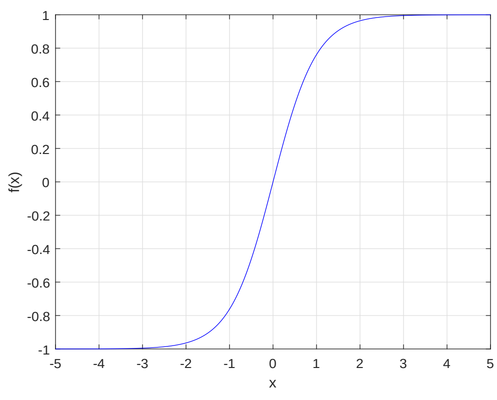
Figure 3.2: A single hidden layer feedforward neural network

Activation functions are predefined, usually nonlinear operations on scalar numbers. There are several common activation functions in the literature, such as the sigmoid, the hyperbolic tangent and the rectifier linear units (ReLU). Figure 3.3 depicts several different activation functions.

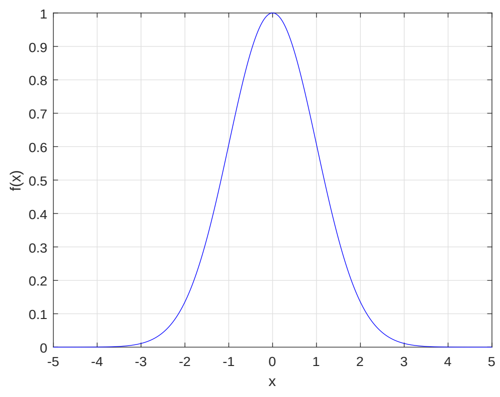
ReLU activations were used throughout this thesis on hidden layers as it has several advantages over the other common activation functions. Krizhevsky et al. found that the convergence of stochastic gradient descent is accelerated by a factor of 6 by using ReLU activations compared to the sigmoid and the hyperbolic tangent [16]. This was attributed to the linear and non-saturating form of the ReLU as opposed to the other two aforementioned activation functions. Another advantage of the ReLU is that it can be represented with a simple, computationally cheap mathematical operation



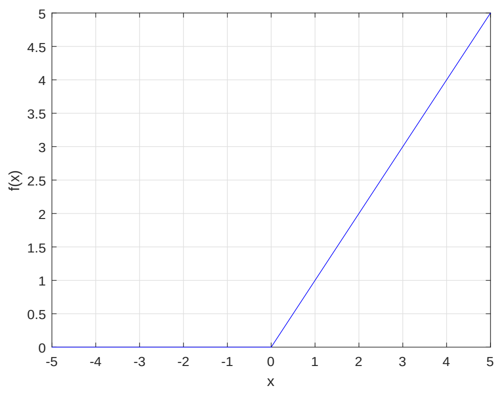
(a) Sigmoid function



(b) Hyperbolic tangent function



(c) Radial basis function



(d) Rectifier linear unit (ReLU)

Figure 3.3: Examples of activation functions

as in Equation 3.11, while the sigmoid and the hyperbolic tangent functions involve expensive operations such as exponentials.

$$f(x) = \max(0, x) \tag{3.11}$$

Activation functions on the output layer of a neural network can also be defined with one of the activation functions given in Figure 3.3 as well as a linear identity function. For classification applications of neural networks sigmoid or hyperbolic tangent activation functions on output layers can be considered, as the outputs of the neural network is usually defined as probabilities of an input set belonging to a class of outputs. For such problems, output is needed to be in the range $o_i \in [0, 1]$ or $o_i \in [-1, 1]$, which can be achieved by sigmoid and hyperbolic tangent activations respectively. On the other hand, for regression problems output is most commonly defined as a vector of arbitrary real-valued numbers with regression targets which are also real-valued. Hence, linear functions were used on the output layer of the neural networks in this thesis, in the form:

$$g(x) = x \tag{3.12}$$

Bias terms v_{l0} and w_{i0} in Equation 3.10 are vectors containing parameters additional to the weights of the neural network. These terms serve the purpose of shifting the output of the activation functions on the input axis, without interfering with the inputs of the neural network. Therefore, In Figure 3.2 outputs of bias nodes are defined as 1.

For a more compact representation of Equation 3.10, outputs of the hidden layer can be written as:

$$z_l = f\left(\sum_{j=1}^{n} v_{lj}x_j + v_{l0}\right) \tag{3.13}$$

where $l = 1, 2, \ldots, L$. Using Equation 3.13, Equation 3.10 can be rewritten as:

$$y_i = g\left(\sum_{l=1}^{L} w_{il}z_l + w_{i0}\right) \tag{3.14}$$

where $i = 1, 2, \ldots, m$. Weights and biases can be represented in matrix form as parameters between the input layer and the hidden layer as:

$$\bar{V} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{L1} & v_{L2} & \cdots & v_{Ln} \end{bmatrix}, b_v = \begin{bmatrix} v_{10} \\ v_{20} \\ \vdots \\ v_{L0} \end{bmatrix} \tag{3.15}$$

Similarly, weights and biases between the hidden layer and the output layer can be represented in matrix form as:

$$\bar{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1L} \\ w_{21} & w_{22} & \cdots & w_{2L} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mL} \end{bmatrix}, b_w = \begin{bmatrix} w_{10} \\ w_{20} \\ \vdots \\ w_{m0} \end{bmatrix} \tag{3.16}$$

Using Equation 3.15 and Equation 3.16, output of the neural network can be written in vector form as:

$$\vec{y} = \vec{g}\left(\bar{W}\vec{f}(\bar{V}\vec{x} + b_v) + b_w\right) \tag{3.17}$$

Bias vectors can be augmented into weight matrices for convenience as defining parameter matrices V and W which contain bias vectors in the first column as:

$$V = \begin{bmatrix} v_{10} & v_{11} & v_{12} & \cdots & v_{1n} \\ v_{20} & v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{L0} & v_{L1} & v_{L2} & \cdots & v_{Ln} \end{bmatrix}, W = \begin{bmatrix} w_{10} & w_{11} & w_{12} & \cdots & w_{1L} \\ w_{20} & w_{21} & w_{22} & \cdots & w_{2L} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m0} & w_{m1} & w_{m2} & \cdots & w_{mL} \end{bmatrix} \tag{3.18}$$

Then, output of the neural network in augmented form becomes:

$$\vec{y} = \vec{g}\left(W\hat{f}\left(V\hat{x}\right)\right) \tag{3.19}$$

where $\hat{x} = [1\, x_1\, x_2\, \ldots\, x_n]^T$ and $\hat{f}$ is the vector of activation functions on hidden layer inputs augmented with 1 as the first entry such that $\hat{f} = \left[1\, \vec{f}^T\right]^T$. Similarly, if Equation 3.13 is written in vector form, one may obtain a more compact form of neural network equations in terms of hidden layer outputs such as:

$$\vec{z} = \hat{f}\left(V\hat{x}\right) \tag{3.20}$$

and

$$\vec{y} = \vec{g}\left(W\vec{z}\right) \tag{3.21}$$

It should be noted that hidden layer output vector $\vec{z}$ is also augmented with 1 as its first entry.

3.3 Backpropagation Method

Reinforcement learning dictates that a learning agent should update its parameters with respect to a reinforcement signal provided by the environment based on the state of the environment and the actions taken by the agent at that particular state. These updates can be achieved if contribution of each network parameter to the error between the output of a neural network and a target value which is expected from that neural network is known. This is called the credit assignment problem. By knowing the contribution of each parameter to the error, one can determine how that parameter can be tuned in order to achieve better outputs. Backpropagation training algorithm was introduced to solve the credit assignment problem [35, 36, 24]. This recursive process of updating the parameters of the agent in the form of a neural network is the standard method of training for reinforcement learning applications. Backpropagation algorithm is a method which can be combined with gradient-based optimization algorithms in order to find updates to the weights of a neural network.

In the scope of this thesis, backpropagation algorithm was employed in order to calculate gradients of a prescribed loss function with respect to the parameters of a neural network. Neural network configuration given in Figure 3.2 will be used again as an example for the derivation of backpropagation method.

Output of the output layer of the neural network configuration given in Figure 3.2 is defined by Equation 3.10, while the hidden layer outputs are described by Equation 3.13. Considering the bias values x_0 and z_0 can be defined as 1 as stated previously for the vector form of the same equations, y_i and z_l can be defined as:

$$y_i = g\left(\sum_{l=0}^{L} w_{il} z_l\right) \tag{3.22}$$

$$z_l = f\left(\sum_{j=0}^{n} v_{lj} x_j\right) \tag{3.23}$$

In addition to the hidden layer outputs, some other intermediate quantities are defined for convenience in order to derive the backpropagation method easily. Let the inputs to the output layer and the hidden layer neurons are respectively u_i^2 and u_l^1 such that

$$u_i^2 = \sum_{l=0}^{L} w_{il} z_l \tag{3.24}$$

$$u_l^1 = \sum_{j=0}^{n} v_{lj} x_j \tag{3.25}$$

Then one can write

$$y_i = g\left(u_i^2\right) \tag{3.26}$$

$$z_l = f\left(u_l^1\right) \tag{3.27}$$

Since backpropagation method is essentially a backward recursive chain rule scheme, partial derivatives of y_i and z_l with respect to quantities appearing prior to them during a forward pass of the neural network are needed. These partial derivatives can be written as:

$$\frac{\partial y_i}{\partial w_{il}} = g'\left(u_i^2\right) z_l \tag{3.28}$$

$$\frac{\partial y_i}{\partial z_l} = g'\left(u_i^2\right) w_{il} \tag{3.29}$$

$$\frac{\partial z_l}{\partial v_{lj}} = f'\left(u_l^1\right) x_j \tag{3.30}$$

$$\frac{\partial z_l}{\partial x_j} = f'\left(u_l^1\right) v_{lj} \tag{3.31}$$

where $f'(.)$ and $g'(.)$ are the derivatives of the activation functions on hidden and output layer neurons respectively.

For the purpose of derivation of the backpropagation algorithm, let weights of the hidden and output layers are updated by the standard gradient descent. Therefore, one can write the updates to the weights as:

$$w_{il} \leftarrow w_{il} + \eta \frac{\partial L}{\partial w_{il}} \tag{3.32}$$

$$v_{lj} \leftarrow v_{lj} + \eta \frac{\partial L}{\partial v_{lj}} \tag{3.33}$$

where L is a cost function, and η is the learning rate. It should be noted that learning rates for the hidden and output layers can be selected differently. In order to update the weights of the network according to Equations 3.32 and 3.33, one must determine the gradients of the cost function L with respect to the weights of the network.

Let inputs to the neural network be an input vector X, and the target output vector for that input vector be Y. If the cost function L is defined as the least-squares error between the target Y and the output vector of the network y, L can be written as:

$$L = \frac{1}{2} \sum_{i=1}^{m} e_i^2 \tag{3.34}$$

where $e_i = Y_i - y_i$.

Gradients of the cost function L can now be determined using the chain rule. Gradients for the weights w_{il} are:

$$\frac{\partial L}{\partial w_{il}} = \frac{\partial L}{\partial u_i^2}\frac{\partial u_i^2}{\partial w_{il}} = \left[\frac{\partial L}{\partial e_i}\frac{\partial e_i}{\partial y_i}\frac{\partial y_i}{\partial u_i^2}\right]\frac{\partial u_i^2}{\partial w_{il}} \tag{3.35}$$

then,

$$\frac{\partial L}{\partial u_i^2} = -g'\left(u_i^2\right)e_i \tag{3.36}$$

$$\frac{\partial L}{\partial w_{il}} = -z_l\left[g'\left(u_i^2\right)e_i\right] \tag{3.37}$$

Gradients for the weights v_{lj} can be similarly obtained as:

$$\frac{\partial L}{\partial v_{lj}} = \frac{\partial L}{\partial u_l^1}\frac{\partial u_l^1}{\partial v_{lj}} = \left[\sum_{i=1}^{m}\frac{\partial L}{\partial u_i^2}\frac{\partial u_i^2}{\partial z_l}\frac{\partial z_l}{\partial u_l^1}\right]\frac{\partial u_l^1}{\partial v_{lj}} \tag{3.38}$$

then,

$$\frac{\partial L}{\partial u_l^1} = -f'\left(u_l^1\right)\sum_{i=1}^{m} w_{il}\left[g'\left(u_i^2\right)e_i\right] \tag{3.39}$$

$$\frac{\partial L}{\partial v_{lj}} = -X_j\left[f'\left(u_l^1\right)\sum_{i=1}^{m} w_{il}\left[g'\left(u_i^2\right)e_i\right]\right] \tag{3.40}$$

The representations of the gradients can be simplified by defining backpropagated errors of the network for hidden and output layers. The backpropagated error for the output layer can be defined as:

$$\delta_i^2 = -\frac{\partial L}{\partial u_i^2} = g'\left(u_i^2\right)e_i \tag{3.41}$$

Similarly, for the hidden layer the backpropagated error is:

$$\delta_l^1 = -\frac{\partial L}{\partial u_l^1} = f'\left(u_l^1\right)\sum_{i=1}^{m} w_{il}\delta_i^2 \tag{3.42}$$

By combining the aforementioned equations for the backpropagation algorithm Equations 3.32 and 3.33 can be rewritten for the parameter updates as:

$$w_{il} \leftarrow w_{il} + \eta z_l \delta_i^2;\ \ i = 1,2,\dots,m;\ \ l = 0,1,\dots,L \tag{3.43}$$

$$v_{lj} \leftarrow v_{lj} + \eta X_j \delta_l^1;\ \ l = 1,2,\dots,L;\ \ j = 0,1,\dots,n \tag{3.44}$$

Backpropagation method can also be transformed to matrix form, which makes computation cheaper for computer programs optimized for matrix operations. In this form, the forward pass of the neural network is given by Equations 3.20 and 3.21, while the backward pass can be described by:

$$e = Y - y \tag{3.45}$$

$$\delta^2 = diag\begin{pmatrix}0 & g'\left(u^2\right)\end{pmatrix} e \tag{3.46}$$

$$\delta^1 = diag\begin{pmatrix}0 & f'\left(u^1\right)\end{pmatrix} W\delta^2 \tag{3.47}$$

where, with u^2 as a vector of size m, $diag(u^2)$ is an $m \times m$ diagonal matrix with entries $u_1^2, u_2^2, \ldots, u_m^2$ on the diagonal. Hence, the parameter updates for the network becomes:

$$W \leftarrow W + \eta z \left(\delta^2\right)^T \tag{3.48}$$

$$V \leftarrow V + \eta X \left(\delta^1\right)^T \tag{3.49}$$

3.4 Optimization

Updates to the parameters θ^q and θ^v were performed by standard noncentered RMSProp algorithm as the optimization method [31]. RMSProp algorithm keeps a running average of the squared gradients for each weight and then divides the gradient by this running average of its recent decayed magnitude. Update rule is given in Equation 3.50:

$$\begin{aligned} g &\leftarrow \alpha g + (1-\alpha)(d\theta)^2 \\ \theta &\leftarrow \theta + \eta \frac{d\theta}{\sqrt{g+\varepsilon}} \end{aligned} \tag{3.50}$$

where g is the moving average of the gradients, α is the decay rate, $d\theta$ is the update on the parameters of the corresponding function approximator, η is the learning rate, and ε is a small constant to avoid division by zero.

CHAPTER 4

PROBLEM FORMULATION

In the previous chapter, key reinforcement learning concepts were described. An actor-critic algorithm that can be used to solve continuous space – continuous action problems such as the helicopter autorotation in a model-free manner, is described in this chapter.

A reinforcement learning task can be described with an environment, an agent which acts upon the environment, and a reward signal emitted by the environment in response to the actions taken by the agent to facilitate learning. This chapter begins with the definition of the environment for the autorotation problem. Chosen reward function and the rationale behind it are explained. Constraints specific to the autorotation dynamics and their incorporation to the reward function are explained. Then, a description of the actor-critic reinforcement learning structure is given. Finally, details of the training process for the agent are presented.

4.1 Definition of Environment

In reinforcement learning context, the environment is defined as the outside world that holds the dynamics of the problem to be solved. For helicopter autorotation, the environment is described by the dynamical system equations given in Chapter 2. Autorotation problem is considered to be solved when touchdown with minimal fuselage kinetic energy is achieved. In light of this, the main component of the reward function is defined as the weighted sums of the squared horizontal velocity and vertical velocity at touchdown.

Acceptable velocities at the instant of touchdown was assumed to be less than $u < 3$ m/s and $w < 1$ m/s. Normalization was applied to the reward function in terms of rotor tip speed, in order to bring the rewards to the order of 1. Objective of the reinforcement learning algorithm is to maximize total obtained reward; hence, minimization of the horizontal and vertical velocities should lead to higher rewards.

Environment states for the reinforcement learning task were also normalized. Normalized environment states are model states given in Equation 2.21. At each discrete time step, the environment receives actions from the agent, marches one step forward in time, and generates next states and a scalar reward. It should be noted that output of the reward function takes zero or negative values only throughout a training episode. Fourth-order Runge-Kutta was used as the numerical approximation method to solve for the ordinary differential equations of the point-mass model.

Actions generated by the agent were chosen to be the normalized changes to the point-mass model inputs, denoted by $\Delta\bar{C_x}$ and $\Delta\bar{C_z}$. This choice of actions were made in order to prevent discontinuities in control inputs to the point-mass model at the instant of engine failure. Note that $\Delta\bar{C_x}$ and $\Delta\bar{C_z}$ were normalized by a factor of 0.01 in order to bring the range of outputs to an order of 1.

4.1.1 Reward Function

Reward function generates a scalar reward at each time step which guides the training of the agent. Reward signal is emitted by the environment and is a function of the state of the environment. Reward functions can be considered as the designer's tuning knobs for reinforcement learning tasks as the desired behavior is defined through reward functions.

Reward function for the autonomous autorotation task in this thesis have a multi-conditional structure. Reward function is defined as a function of height h, vertical velocity w, thrust coefficient over rotor solidity C_T/σ, rotor angular speed Ω, and rotor disk orientation α.

Reward component with respect to height h is defined as:

$$r_h = \begin{cases} \frac{-100(u^2+3w^2)}{(\Omega_0 R)^2} & \text{if } h >= 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Since acceptable touchdown velocity for vertical velocity is assumed to be one-third of the horizontal velocity, a weight of 3 was multiplied with squared vertical velocity. Multiplication with $100/(\Omega_0 R)^2$ was performed for normalization. Negative sign of the reward is due to the nature of reinforcement learning where the rewards are maximized; hence, maximum reward from this component corresponds to minimum vertical and horizontal velocities. It should be noted that r_h takes a negative value only at the instant of touchdown. Since $h >= 0$ is also a terminal condition, r_h is obtained once per episode. Note also that h is negative above the ground.

Vertical velocity component of the reward function, i.e. r_w is calculated as:

$$r_w = \begin{cases} \frac{10-w}{1000} & \text{if } w > 10 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Vertical velocity above 10 m/s is punished in order to prevent rotor overspeeding usually observed with high vertical speeds. This definition of reward function component ensures that the agent receives small negative rewards at each time step vertical velocity is above 10 m/s.

Thrust coefficient over rotor solidity, or rotor loading, term of the reward is defined as:

$$r_{(C_T/\sigma)} = \begin{cases} 0.15 - \frac{C_T}{\sigma} & \text{if } \frac{C_T}{\sigma} > 0.15 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Lee et al. stated that stall limit for the rotor loading is 0.15 for the OH-58A helicopter [18]. After reaching this limit, profile drag coefficient of the rotor increases. This is a major contributing factor of rotor angular speed loss and should be avoided. To ensure that rotor loading remains below stall limit, a large penalty is applied to the agent at every time step where stall limit is exceeded.

Rotor angular speed also contributes to the reward function in a negative fashion in case it is outside predefined limits, such that:

$$r_\Omega = \begin{cases} \frac{1.1\Omega_0 - \Omega}{1000} & \text{if } \Omega > 1.1\Omega_0 \\ \frac{\Omega - 0.7\Omega_0}{1000} & \text{if } \Omega < 0.7\Omega_0 \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

Rotor speed Ω is not desired to operate outside limits defined in Equation 4.4. To ensure that the agent learns not to drive the point-mass model above or below these rotor speed limits, small penalties were associated with states that violates these limits.

Finally, reward function has a component corresponding to the rotor disk orientation α:

$$r_\alpha = \begin{cases} \frac{30 - \alpha}{2000} & \text{if } \alpha > 30 \\ \frac{\alpha + 30}{2000} & \text{if } \alpha < -30 \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Rotor disk orientation α is desired to remain in the range $[-30, 30]$ to prevent excessive maneuvering during autorotation, and values of α outside these limits were penalized by a small negative reward.

Total reward is the sum of all terms mentioned above, such that:

$$r = r_h + r_w + r_{(C_T/\sigma)} + r_\Omega + r_\alpha \tag{4.6}$$

4.2 Agent Structure

The most significant part of the reinforcement learning task is the agent, as it is the entity that infers the necessary actions to take to achieve an objective, given the environment states. The agent consists of two neural network function approximators, namely the critic network and the actor network. Actor-critic system architecture is depicted in Figure 4.1

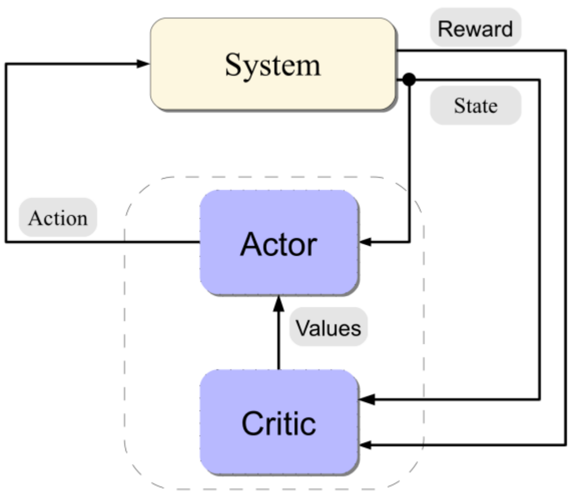


Figure 4.1: Actor-Critic Architecture

4.2.1 The Critic Network

The critic network is a neural network function approximator with two hidden layers, where each layer consists of 32 neurons. Inputs to the critic network are normalized point-mass model states, concatenated with normalized control variables such that $s = \left[\bar{w}\,\bar{u}\,\bar{\Omega}\,\bar{h}\,\bar{C}_x\,\bar{C}_z\right]^T$. Output of the value function is a scalar $V(s)$, which is the value estimation of the state set defined by the inputs to the network. Critic network's output $V(s)$ can be considered as an approximation to the expected total reward from a given state set following a policy π. Parameters of the critic network including both weights and biases are denoted as θ^v throughout this chapter. Therefore, critic network is denoted as $V(s;\theta^v)$.

4.2.2 The Actor Network

The actor network is identical to the critic network in terms of structure, except that its output is a vector of real numbers, namely $\mu(s;\theta^q) = \left[\mu_{C_x}\,\mu_{C_z}\right]^T$. Output of the actor network is defined as an input to the Gaussian distribution as the mean value in order to determine the output of the policy $\pi(s;\theta^q)$, where θ^q are the parameters of the actor network. To clarify, actions produced by the actor network are sampled from

Gaussian distributions with mean values as actor network's outputs, such that:

$$\pi(s;\theta^q) = \left[\Delta\bar{C_x}\,\Delta\bar{C_z}\right]^T \tag{4.7}$$

where $\Delta\bar{C_x}$ and $\Delta\bar{C_z}$ are sampled from:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.8}$$

where σ is the standard deviation and μ as mean or center produced by the actor network. This random sampling from Gaussian distribution is required to facilitate exploration of action space by the agent. Reinforcement learning algorithm which will be presented next requires exploration to obtain the optimal policy which achieves the maximum reward starting from any state s from the state space.

4.3 Training Algorithm

Reinforcement learning algorithm used in this thesis follows the conventional terminology of the broader reinforcement learning field. At each discrete time step t, the agent observes the state of the environment s_t and executes an action a_t depending on s_t and its current policy function π, while the environment receives the action a_t and move to a new state s_{t+1}. The agent then receives a scalar reward r_t, which measures the success of a policy at that time step. Policy π describes the mapping of states s_t to actions a_t as conceived by the agent. The aim of the general reinforcement learning framework is to learn a policy from observations which maximizes the total reward starting from any state s_t.

The algorithm used here is a model-free on-policy method which falls under the category of policy-based methods, and it is a simplified version of the Asynchronous Advantage Actor-Critic (A3C) algorithm introduced in [22]. Policy-based reinforcement learning focuses on estimating the optimal policy π^*, which is the policy maximizing the future reward. The algorithm uses an actor-critic architecture, where the actor is considered as the policy function $\pi(s_t;\theta^q)$ and the critic is an approximation to the expected total reward from a given state s_t following the policy π, which is defined as the value function $V(s_t;\theta^v)$. In this study, actor and critic were represented as function approximators in the form of separate neural networks with θ^q and θ^v as parameters (or weights) of the networks respectively.

An important feature of the algorithm is the asynchronous execution of training. A multi-core CPU was used for instances of training episodes. Each thread on the CPU runs a worker process which performs training with its own copy of the actor network, the critic network, the environment, and the numerical solver. Parallelization of the training allows for different exploration policies for different threads; however, a fixed exploration policy was used across all training threads here. For each episode, a worker process initializes its own copy of the environment, and samples action a_t from a Gaussian distribution with mean value as the output of the actor network and standard deviation $\sigma = 1$. The environment then marches one step forward in time and generates the reward achieved for that step. The action a_t, the state s_t and the reward r_t at each time step are stored in an array on memory. Upon reaching a terminal state, the training episode ends and gradients necessary for updating network parameters θ^q and θ^v are calculated via backpropagation using the stored a_t, s_t, and r_t at each time step. Gradients then summed up into a total gradient update for that single episode. These accumulated gradients are sent to a parent process to update the main actor and critic networks. Updated actor and critic networks are shared across each new worker process on each thread in subsequent training episodes. This way of updating the function approximator parameters is shown to have a stabilizing effect on training without relying on experience replay methods, which require large memory spaces [22]. As mentioned before, updates to the parameters θ^q and θ^v were handled by RMSProp optimization. RMSProp is an improved version of standard gradient descent optimization, which facilitates faster convergence and avoids potential plateaus of loss functions by using the so called momentum of previous updates. Complete algorithm used in this study is described in Algorithm 1 in Appendix.

4.3.1 Network Parameter Updates

In this section, calculation of updates to the parameters of actor and critic networks are explained. Details and rationale behind chosen update rules are given for both networks.

4.3.1.1 Critic Network Updates

Aim of updating the critic network parameters is to train the critic network to give better estimations of the values of all state sets in the state space. Therefore, the loss function to be minimized for critic network updates takes the form as follows:

$$J_{critic,t} = \frac{1}{2}(R_t - V(s_t;\theta^v))^2 \tag{4.9}$$

where R_t is the discounted total reward from time step t until the end of the training episode, such that:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ... = r_t + \gamma R_{t+1} \tag{4.10}$$

For the minimization of the cost $J_{critic,t}$, gradients of the cost function with respect to the critic network parameters are needed. From chain rule:

$$\begin{aligned} \frac{\partial J_{critic,t}}{\partial \theta^v} &= \frac{\partial J_{critic,t}}{\partial V(s_t;\theta^v)} \frac{\partial V(s_t;\theta^v)}{\partial \theta^v} \\ &= -(R_t - V(s_t;\theta^v))\frac{\partial V(s_t;\theta^v)}{\partial \theta^v} \end{aligned} \tag{4.11}$$

where partial derivative of critic network output $V(s_t;\theta^v)$ with respect to the network parameters θ^v were calculated by backpropagation as detailed in Section 3.3. As a result, update rule for the parameters of the critic network becomes:

$$d\theta^v \leftarrow d\theta^v + (R_t - V(s_t;\theta^v))\frac{\partial V(s_t;\theta^v)}{\partial \theta^v} \tag{4.12}$$

4.3.1.2 Actor Network Updates

Similar to the rationale of training the critic network, training of the actor network aims to find a mapping from states to actions that maximizes total reward starting from any state s_t. The loss function to be minimized for actor network updates is:

$$J_{actor,t} = \frac{1}{2}(a_t - \mu(s_t;\theta^q))^2 \tag{4.13}$$

where a_t is the sampled action from policy π. Gradients of this cost function with respect to the actor network parameters were calculated as:

$$\begin{aligned} \frac{\partial J_{actor,t}}{\partial \theta^q} &= \frac{\partial J_{actor,t}}{\partial \mu(s_t;\theta^q)} \frac{\partial \mu(s_t;\theta^q)}{\partial \theta^q} \\ &= -(a_t - \mu(s_t;\theta^q)) \frac{\partial \mu(s_t;\theta^q)}{\partial \theta^q} \end{aligned} \tag{4.14}$$

Gradients defined by Equation 4.14 do not serve the purpose of training the network towards more suitable actions alone. In fact, using Equation 4.14 only would lead to actions trained randomly for each state. To prevent this, updates are calculated such that:

$$d\theta^q \leftarrow d\theta^q + (a_t - \mu(s_t;\theta^q)) \frac{\partial \mu(s_t;\theta^q)}{\partial \theta^q} (R_t - V(s_t;\theta^v)) \tag{4.15}$$

Error between the real value and estimated value of a state, i.e. $R_t - V(s_t;\theta^v)$, provides the knowledge of when a better or worse than expected state is achieved through an action. Hence, a positive error term means that actions which yielded better rewards are reinforced, and a negative error means that actions performed worse than expected are avoided. In other words, better than expected states push the outputs of the actor value (center points of the Gaussian distribution) towards the sampled actions from corresponding Gaussian distribution, which in turn makes those sampled actions more likely to be taken by the agent in the future. Conversely, worse than expected states push actor outputs away from sampled actions.

4.3.2 Hyperparameters of The Training Algorithm

Most of the hyperparameters of the training algorithm were found by trial-and-error. Selected parameter values are: Time step for training episodes $d_t = 0.1$, maximum number of training episodes $N_{max} = 2 * 10^6$, discount factor $\gamma = 0.99$, actor learning rate $\eta_{actor} = 0.0003$, critic learning rate $\eta_{critic} = 0.00015$, RMSProp decay rate $\alpha_d = 0.99$ and RMSProp constant $\varepsilon = 0.1$.

Learning rates of both critic and actor networks were annealed linearly with episode count so that when $n = N_{max}$ both learning rates approach zero.

CHAPTER 5

SIMULATION RESULTS

Autorotation problem was formulated by utilizing a point-mass model and a reinforcement learning agent. Then, the agent was trained in order to solve the problem by minimizing the horizontal and vertical velocities of the point mass model at touchdown; hence, kinetic energy at the instant of ground impact is minimized. States of the point mass mathematical model was the vertical velocity w, the horizontal velocity u, the rotor angular speed Ω, the height above the ground level h, and the horizontal distance from the point of engine failure x. Since this study is not concerned with landing the helicopter to a predefined landing spot, there are no constraints on the horizontal distance state x. Therefore, x is not an input to the reinforcement learning agent. Inputs to the model were defined as the horizontal thrust coefficient C_x and the vertical thrust coefficient C_z.

Inputs to the reinforcement learning agent consists of model inputs, in addition to the model states except the horizontal distance state, while outputs of the agent, i.e. actions, were selected as changes to the horizontal and vertical thrust coefficients at each time step, namely ΔC_x and ΔC_z. This choice of actions was preferred; because, it prevents discontinuities in model control inputs C_x and C_z at the instant of engine power loss.

Standard OH-58A helicopter has an Height-Velocity curve knee for 1360 kg gross weight and a rotor Lock number of 5.43 as depicted in Figure 5.1. Low altitude region which is also the part of standard flight manuals is not included here, since it depicts a zone where the pilots do not feel comfortable during flare phase due to the proximity of the tail boom to the ground and there is a risk of contact between the tail and the ground. Because the study presented in this thesis is conducted with a point-mass model of a helicopter in autorotation, this low speed region of the H-V diagram is irrelevant. Inside the low-speed knee region of the depicted diagram, safe landing through autorotation is not guaranteed by the helicopter manufacturer.

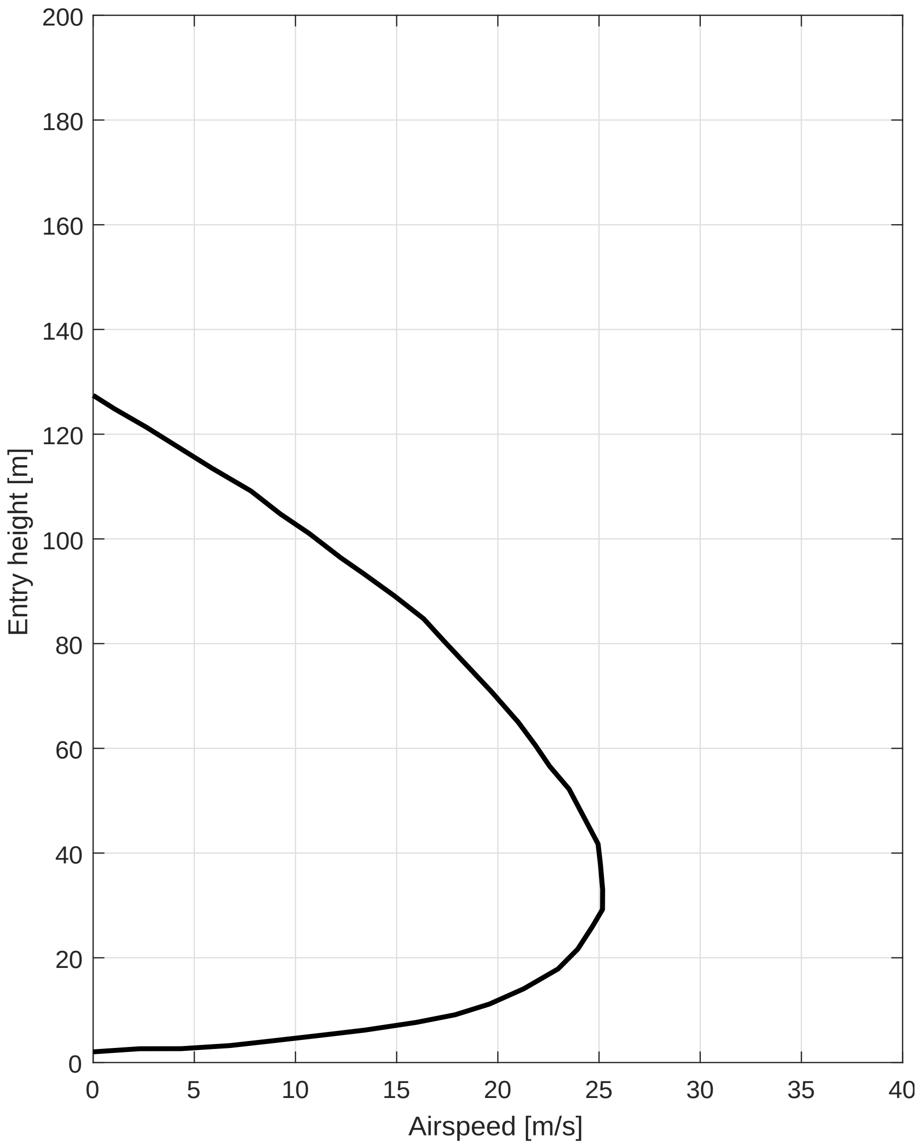


Figure 5.1: OH-58A H-V diagram for Gross Weight = 1360 kg, Lock number = 5.43

It should be noted that hover condition is not included in the training algorithm and the resulting agent was not tested for autorotation starting from hover. This is due to the fact that optimal trajectory for autorotation from hover is purely vertical, and a more meaningful training algorithm could be set up by omitting horizontal velocity state u, and horizontal thrust coefficient input C_x from the dynamical equations. This approach would reduce the problem to one-dimensional motion in vertical plane and would naturally find trajectories which are purely vertical. Optimal trajectories for vertical autorotation are investigated in [14].

Various conditions were selected from the regions both inside and outside the knee of the H-V diagram in order to investigate the agent's performance against the effects of autorotation entry height and entry speed. Selected test conditions are given in Table 5.1 and plotted against the H-V diagram in Figure 5.2.

Table 5.1: Autorotation Entry Conditions

Entry Condition	1	2	3	4	5	6
Height [m]	40	40	60	175	90	40
Airspeed [m/s]	5	20	2	5	5	30

Test conditions 1, 5, and 4 were selected to investigate the agent's performance during autorotation from different entry heights for a forward velocity of 5 m/s. Entry heights for these conditions are 40 m, 90 m, and 175 m respectively. Similarly, test conditions 1, 2, and 6 were used for observing the effect of different autorotation horizontal entry speeds from the same entry height of 40 m. For these test conditions entry speeds are 5 m/s, 20 m/s, and 30 m/s in ascending order. Test condition 3 was selected in order to be able to investigate a near-hover entry condition to the autorotation. Therefore, forward speed for test condition 3 was selected as 2 m/s. All test conditions were at steady-state level flight condition before the engine power loss.

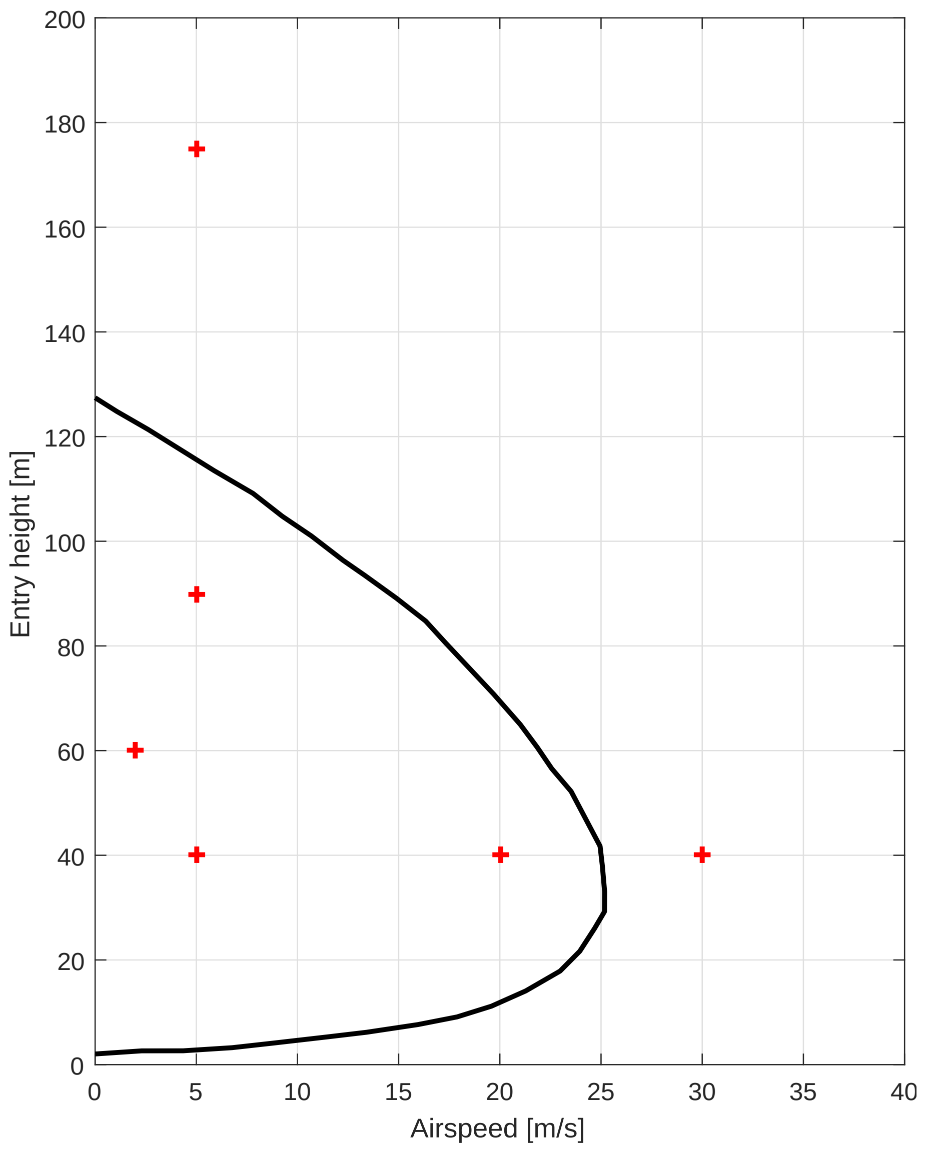
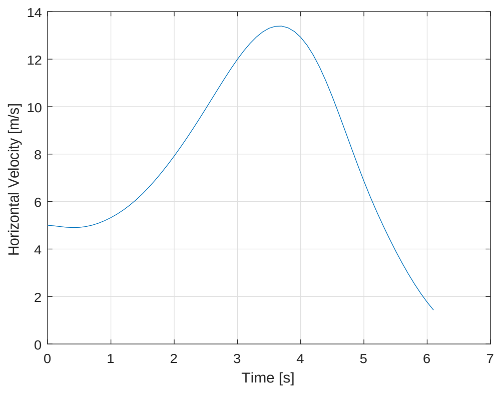


Figure 5.2: Test points on OH-58A H-V diagram

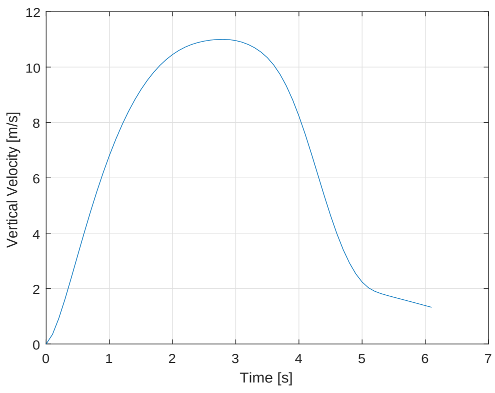
5.1 Interpretation Of The Results

A detailed presentation of the results achieved by the reinforcement learning agent can be found in Figure 5.3 for an autorotation maneuver from an entry height of 40 m and at 5 m/s initial forward speed. This test case corresponds to the test condition 1 in Table 5.1.

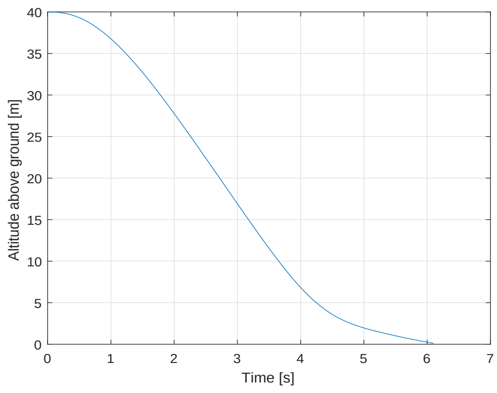
Initially, the collective was decreased by the agent in order to arrest the rotor speed decay. Forward velocity was increased from 5 m/s, since the speed of minimum required power is above the initial speed. An important result of this study is that the



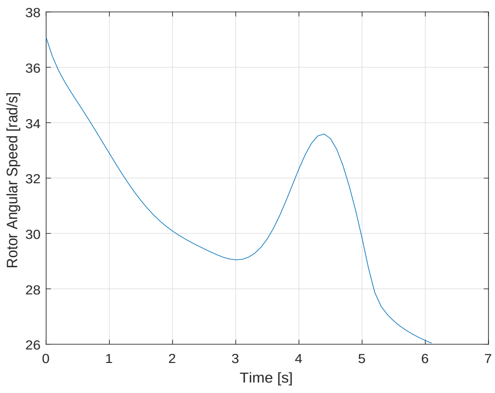
(a) Time variation of horizontal velocity u



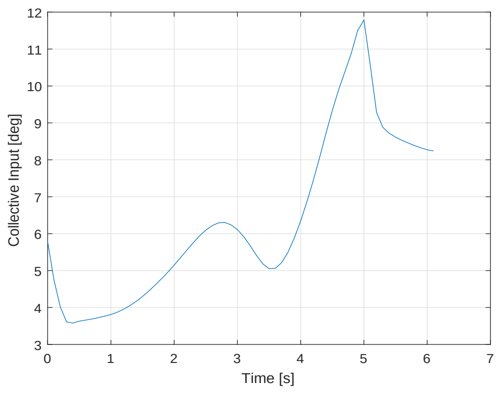
(b) Time variation of vertical velocity w



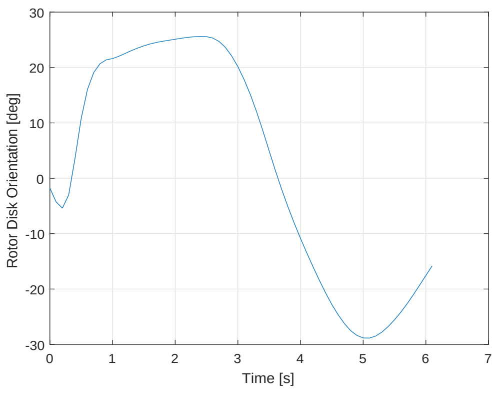
(c) Time variation of altitude h



(d) Time variation of rotor angular speed Ω



(e) Time variation of collective control



(f) Time variation of rotor disk angle

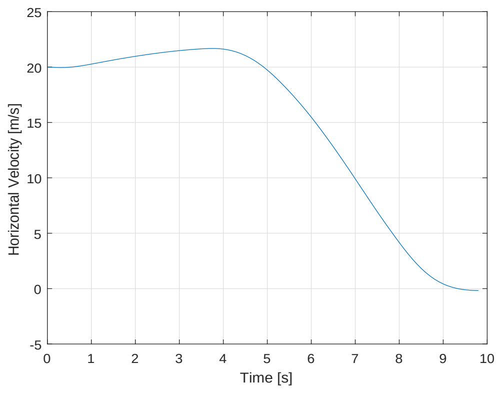
Figure 5.3: Autorotation trajectory for test condition 1

agent learns to find an optimal forward speed during autorotation at which the descent is steady and sink-rate can be held constant at a reasonable value. Due to the actions taken above, initial response of the point-mass model is an increase in vertical velocity up to a maximum of 11 m/s. However, this increase of vertical velocity is necessary in order to maintain rotor angular speed. The agent then executes a flare maneuver by a rearward tilt of the rotor disk, while simultaneously increasing collective in order to prevent the rotor from overspeeding. Flare maneuver reduces both the forward speed and the vertical speed of the helicopter to near zero values. From that point on, the agent uses the remaining energy in the rotor for a cushioned landing by further increasing the collective. This energy extraction from the main rotor can be observed from the rotor speed decay after the flare phase of the maneuver. At the instant of touchdown horizontal velocity is below 3 m/s; however, the agent did not succeed in keeping a touchdown vertical velocity below 1 m/s. Hence, the agent could not meet the success criteria and this test condition corresponds to a failed landing.

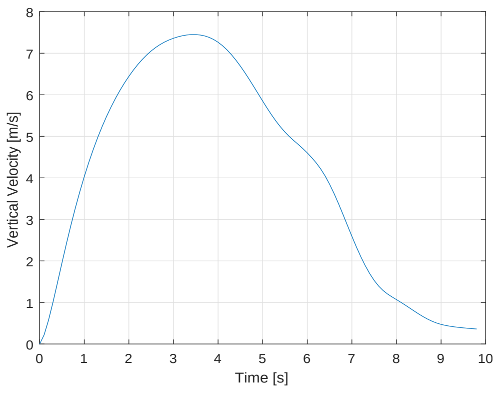
Trajectories for the test condition 2 are given in Figure 5.4. This test has an entry height of 40 m, and an entry speed of 20 m/s.

Similar to the test condition 1, collective input is decreased as the first response to the engine loss at $t = 0$. However, the agent managed to maintain the rotor speed and vertical velocity at a limited range without needing a sharp drop in collective, as depicted in Figure 5.4e. Horizontal velocity kept almost constant until the flare at $t = 4$ seconds. Rotor disk angle was close to 30 degrees backwards at maximum tilt. The agent managed to touch the ground with zero rotor disk angle, unlike the previous condition. Collective input trend is similar to the test condition 1; however, maximum collective used throughout the maneuver is less, which is due to the fact that the initial condition of horizontal velocity is close to the maximum endurance speed of the helicopter. Finally, touchdown was performed at zero horizontal velocity and vertical velocity is below 1 m/s; therefore, it can be concluded that the agent was successful in landing the helicopter for test condition 2.

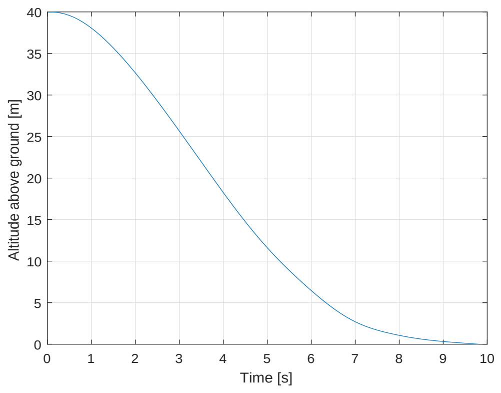
Trajectories of remaining test conditions are given in Figures 5.5 to 5.8.



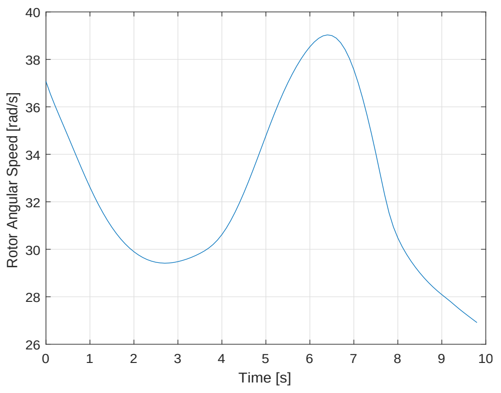
(a) Time variation of horizontal velocity *u*



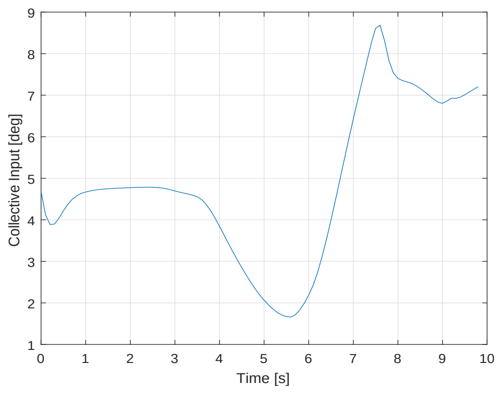
(b) Time variation of vertical velocity *w*



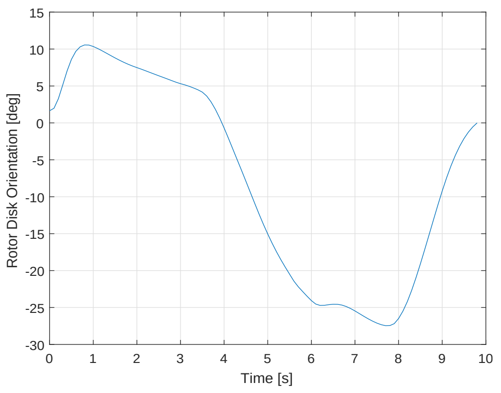
(c) Time variation of altitude *h*



(d) Time variation of rotor angular speed Ω

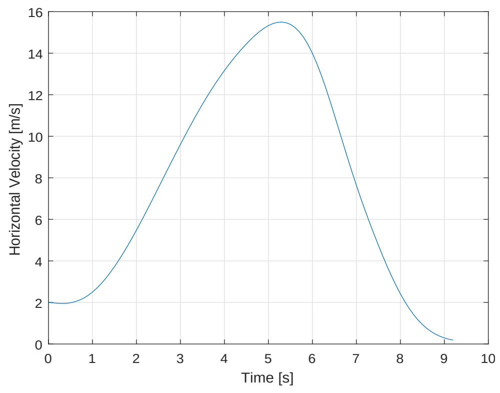


(e) Time variation of collective control

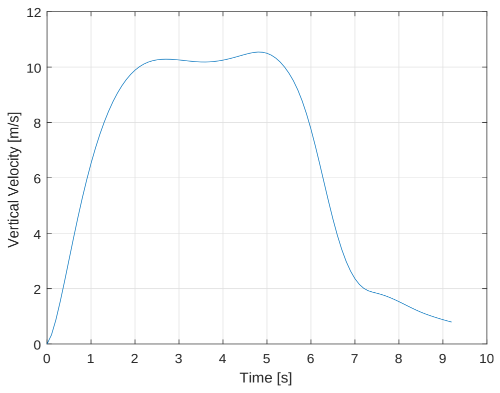


(f) Time variation of rotor disk angle

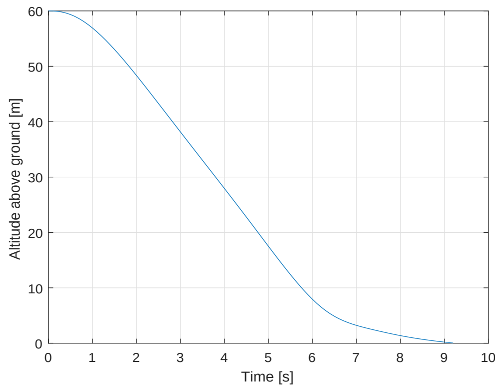
Figure 5.4: Autorotation trajectory for test condition 2



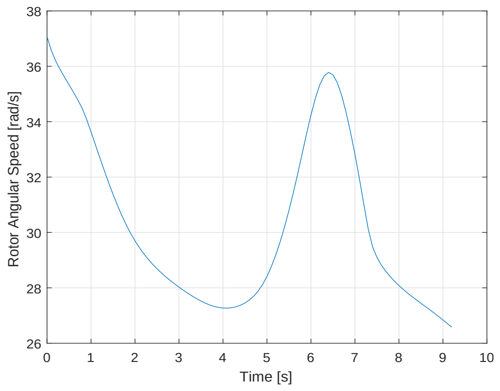
(a) Time variation of horizontal velocity *u*



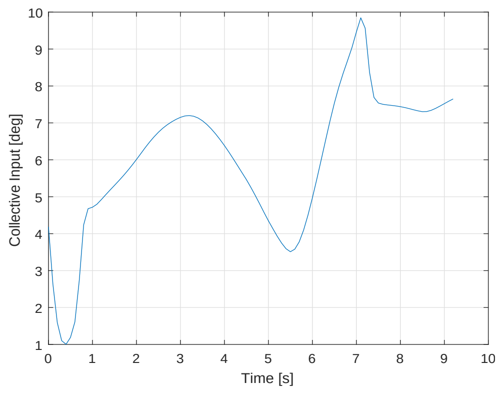
(b) Time variation of vertical velocity *w*



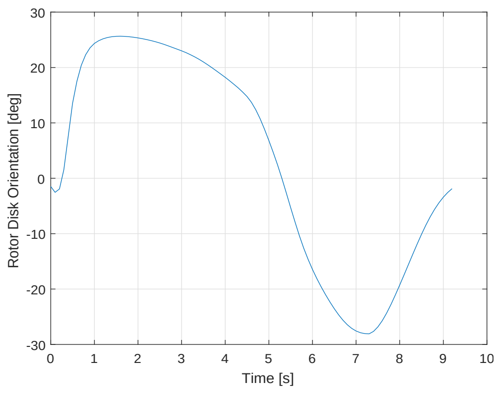
(c) Time variation of altitude *h*



(d) Time variation of rotor angular speed Ω

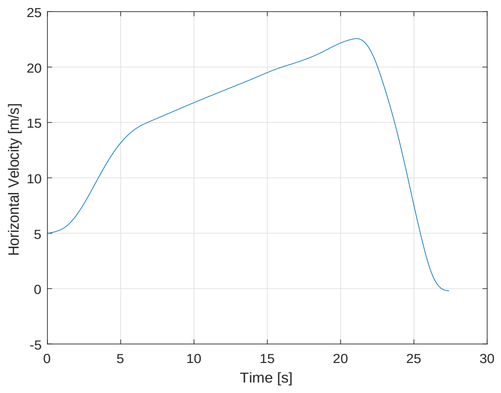


(e) Time variation of collective control

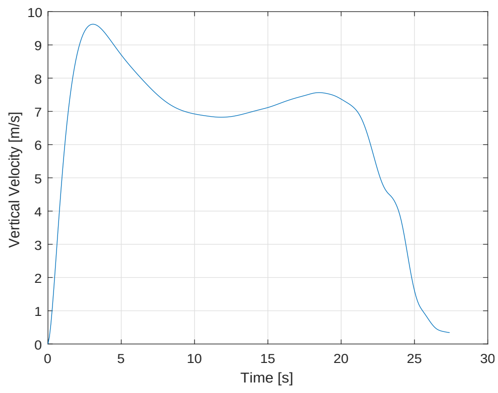


(f) Time variation of rotor disk angle

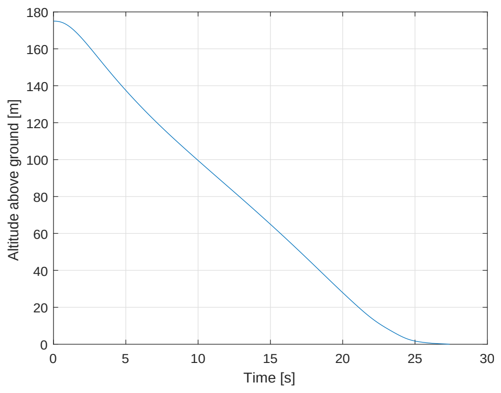
Figure 5.5: Autorotation trajectory for test condition 3



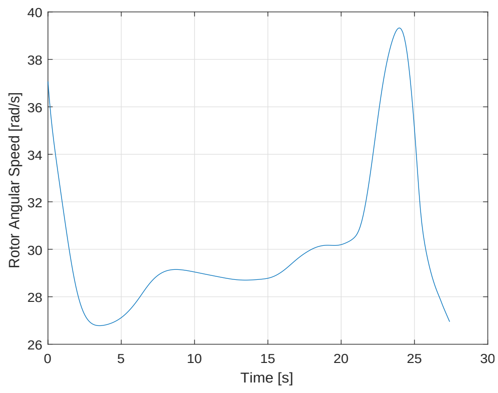
(a) Time variation of horizontal velocity u



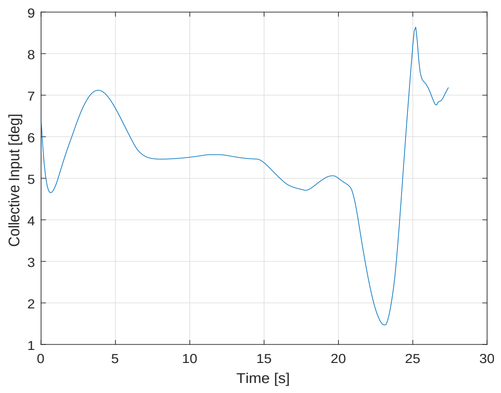
(b) Time variation of vertical velocity w



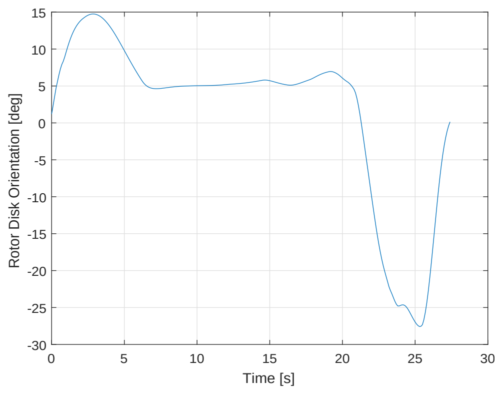
(c) Time variation of altitude h



(d) Time variation of rotor angular speed Ω

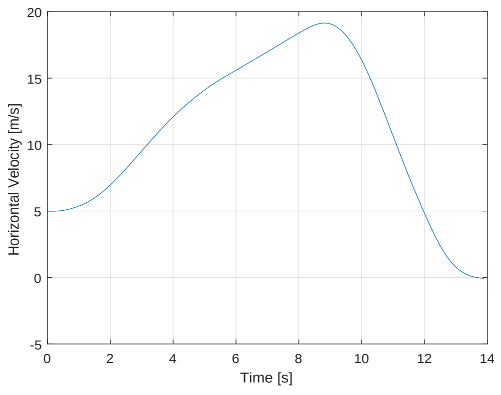


(e) Time variation of collective control

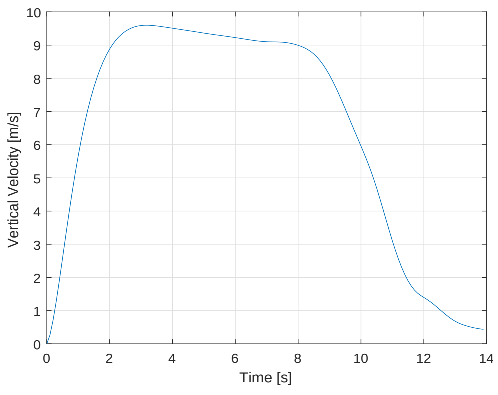


(f) Time variation of rotor disk angle

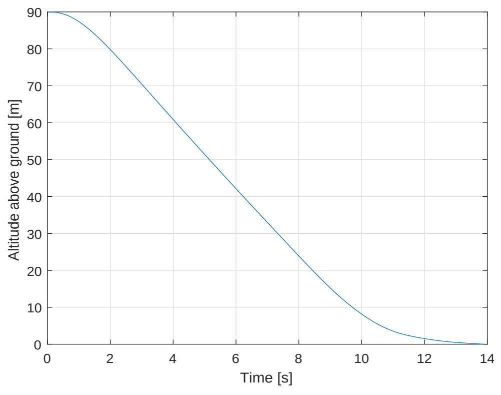
Figure 5.6: Autorotation trajectory for test condition 4



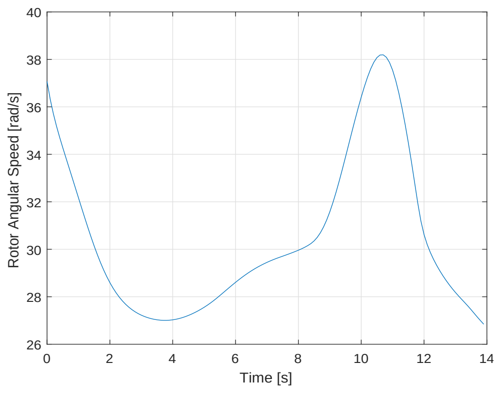
(a) Time variation of horizontal velocity u



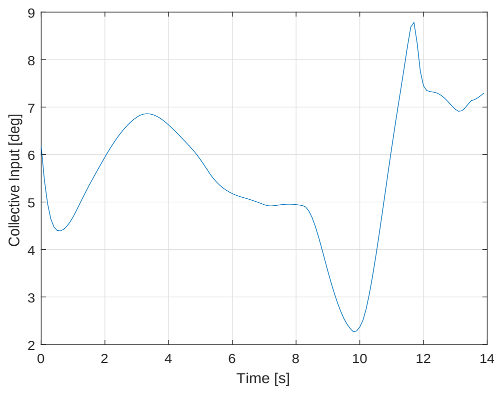
(b) Time variation of vertical velocity w



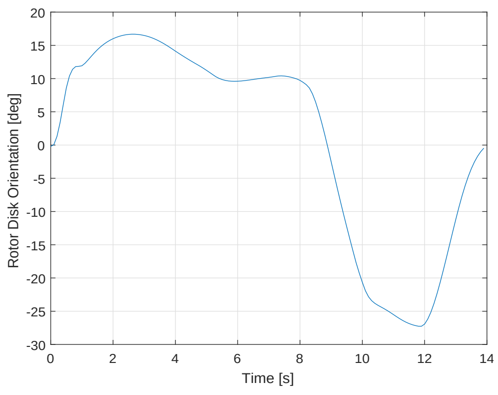
(c) Time variation of altitude h



(d) Time variation of rotor angular speed Ω

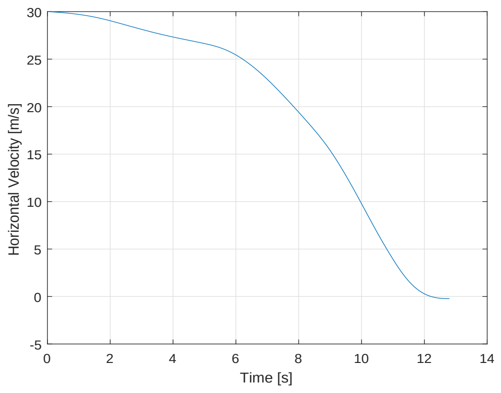


(e) Time variation of collective control

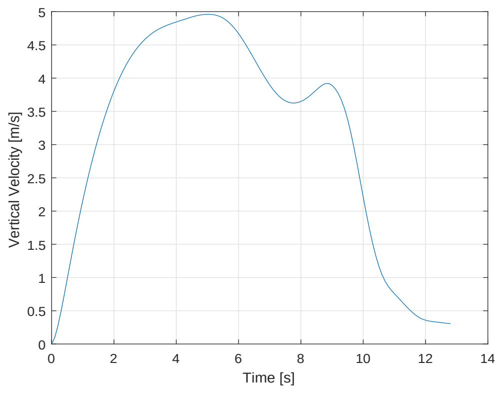


(f) Time variation of rotor disk angle

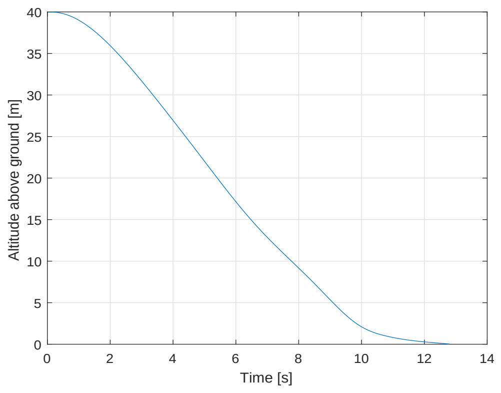
Figure 5.7: Autorotation trajectory for test condition 5



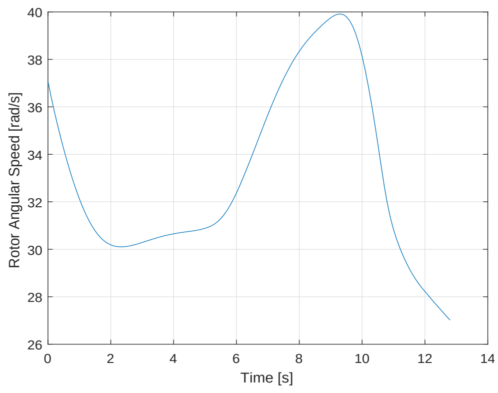
(a) Time variation of horizontal velocity u



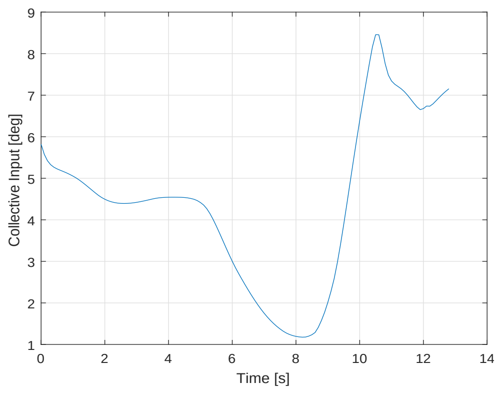
(b) Time variation of vertical velocity w



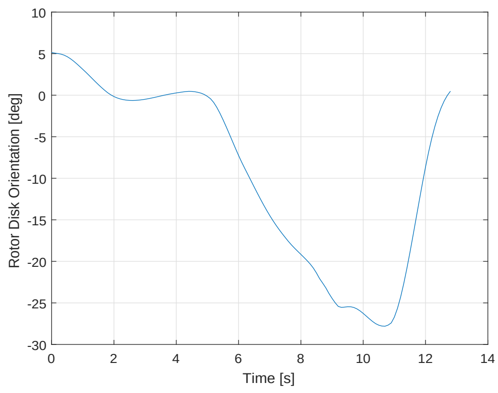
(c) Time variation of altitude h



(d) Time variation of rotor angular speed Ω



(e) Time variation of collective control



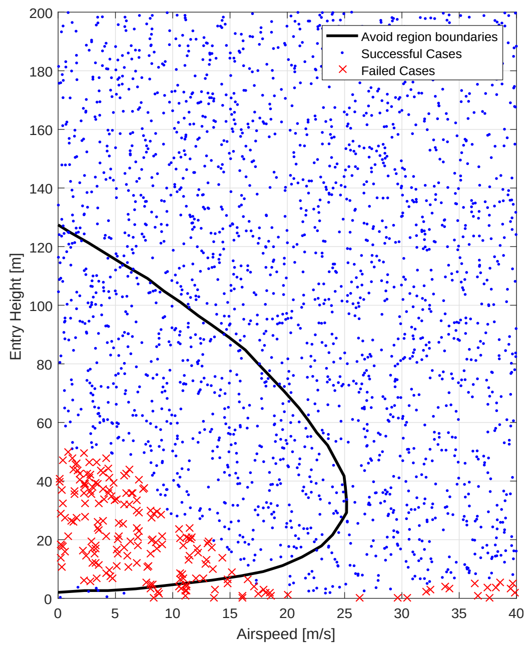
(f) Time variation of rotor disk angle

Figure 5.8: Autorotation trajectory for test condition 6

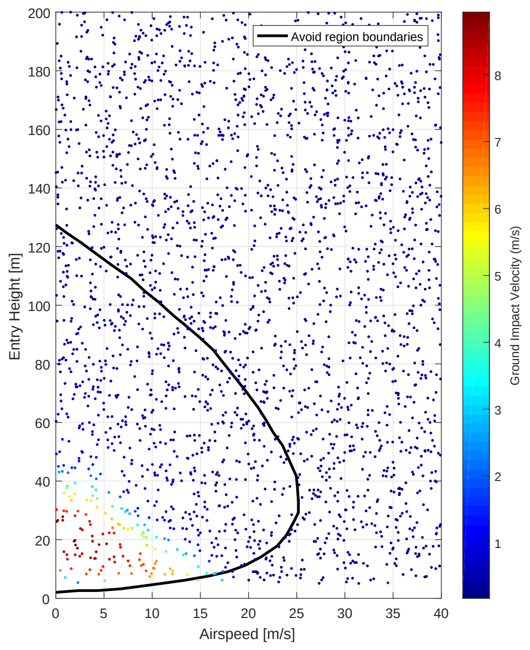
5.2 Monte Carlo Simulations of Random H-V Initial Conditions

Some success criteria must be defined to measure the ability of the trained agent to safely land the helicopter after engine power loss. A reasonable measure for horizontal speed can be defined as $u < 3$ m/s, while the vertical velocity limit for successful touchdown was taken as $w < 1$ m/s. The vertical speed criteria used here is considered stringent, as Lee et al. used approximately $w < 2.5$ m/s as their success criteria for the same helicopter [18].

After training is completed, the agent was tested against random initial conditions both inside and outside H-V curves for OH-58A at different gross weights of 1360 kg, 1250 kg, 1100 kg and 1000 kg. These tests show that the agent can successfully land the point-mass helicopter model in autorotation for the majority of the test points for all gross weight parameters. Results given in Figures 5.9 to 5.12 show that number of random test cases which result in a failed landing according to the aforementioned success criteria decreases with decreasing gross weight. It is evident from the results that unsuccessful test cases appear mostly at the low total energy region at autorotation entry, which is not surprising since there is more available energy for a successful autorotative landing for high altitude and/or high speed entry points. Total speed achieved by the agent at touchdown with respect to the entry points were also given for different gross weight configurations. Similar to the successful/failed landing results, touchdown speeds were observed to be smaller for decreasing gross weight, and low total energy at autorotation entry leads to higher touchdown speeds.

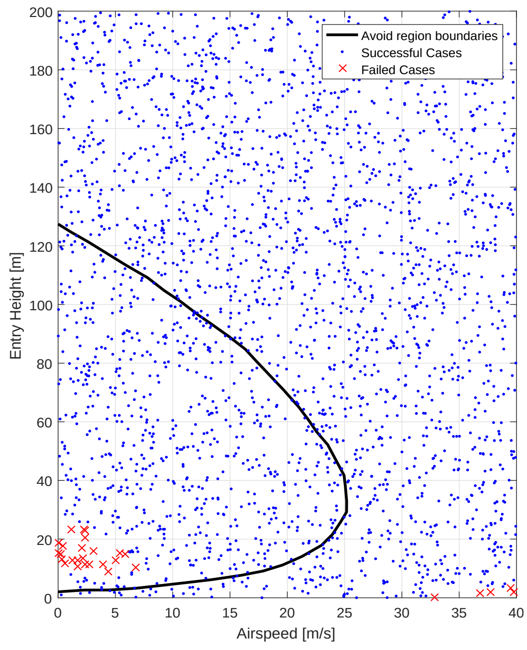


(a) Successful/failed cases wrt. entry points

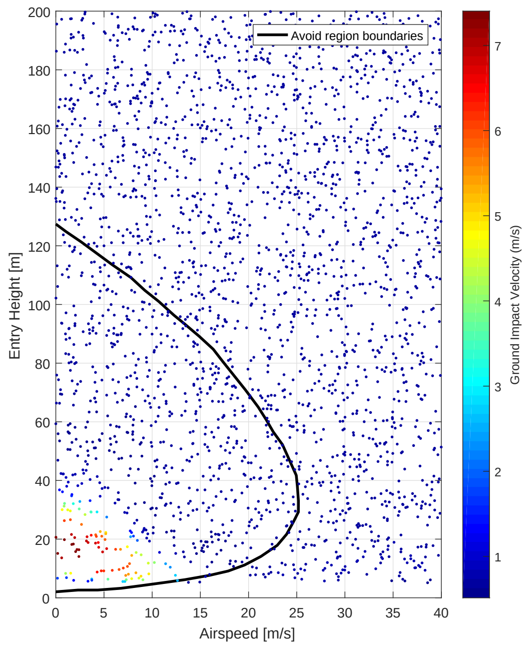


(b) Touchdown speed wrt. entry points

Figure 5.9: Monte Carlo simulation results of random autoration entry points for $m = 1360$ kg

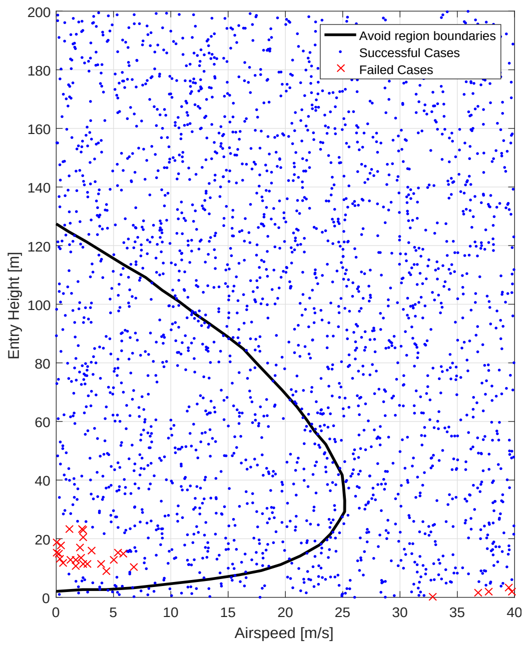


(a) Successful/failed cases wrt. entry points

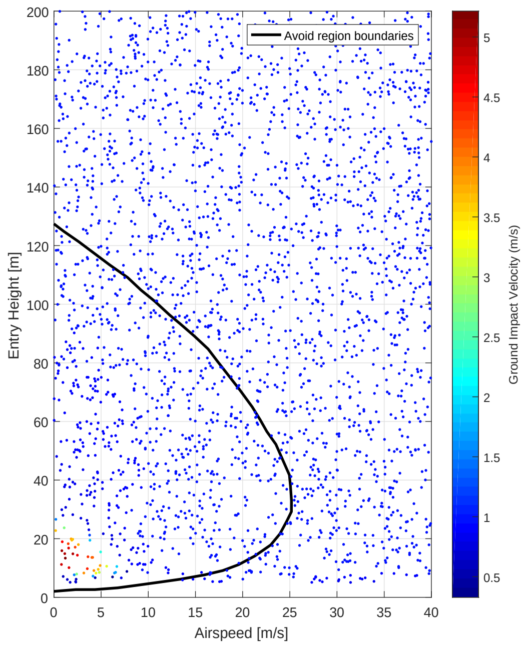


(b) Touchdown speed wrt. entry points

Figure 5.10: Monte Carlo simulation results of random autoration entry points for $m = 1250$ kg

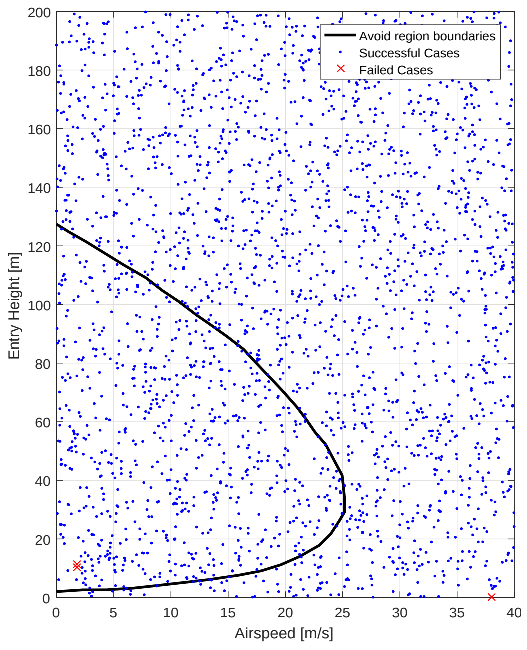


(a) Successful/failed cases wrt. entry points

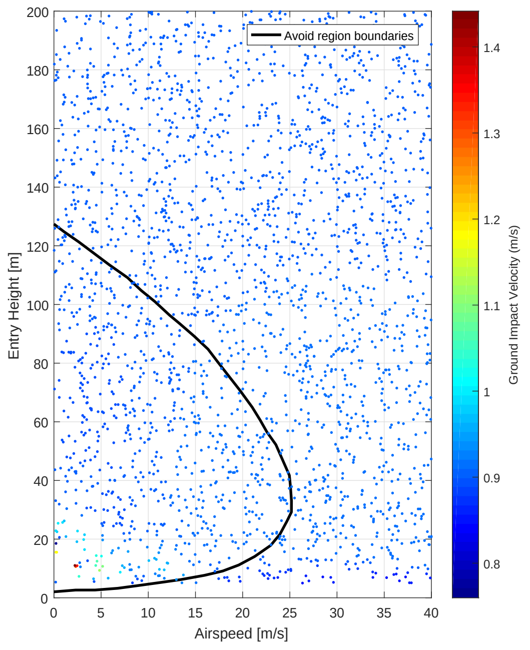


(b) Touchdown speed wrt. entry points

Figure 5.11: Monte Carlo simulation results of random autoration entry points for $m = 1100$ kg



(a) Successful/failed cases wrt. entry points



(b) Touchdown speed wrt. entry points

Figure 5.12: Monte Carlo simulation results of random autoration entry points for $m = 1000$ kg

5.3 Effect of Number of Training Episodes on Agent Performance

Total number of training episodes during reinforcement learning plays an important factor on the agent performance. To demonstrate the effect of number of training episodes on the agent's behaviour after training is completed, a series of simulations were conducted with the point-mass model at $m = 1360$ kg gross weight. Simulations were run for 2500 randomly selected initial condition points for autorotation entry height and entry speed. It should be noted that agents trained for varying number of training episodes were tested against the same randomly chosen initial condition set. Random initial conditions for entry height was selected such that $h_0 \in [0, 200]$ meters. Initial entry speeds were chosen from $u_0 \in [0, 40]$ meters per second. To show how the agent behaves at early stages of learning, the agent's performance were measured after the first five multiples of 100K training episode intervals. After that, measurements were taken at every multiple of 500K training episodes for a total number of episodes of 2 million. Performance of the agent after each interval were measured in terms of 1) the ratio of the number of successful landing cases to all cases starting from the initial condition set according to the criteria given in Section 5.2, and 2) average of the total touchdown speed achieved by the agent for each initial condition. Results were given in Table 5.2. Results show that training for more episodes increase the success rate of the agent up to 94%, while reducing average touchdown speed to roughly 0.5 m/s.

Table 5.2: Agent performance with varying training intervals

Number of episodes	% of successful landings	Average touchdown speed [m/s]
100K	2.28	2.13
200K	84.44	1.39
300K	89.80	1.11
400K	90.32	0.93
500K	92.44	0.83
1M	94.40	0.72
1.5M	94.08	0.62
2M	94.52	0.56

CHAPTER 6

CONCLUSION

In this thesis, a reinforcement learning approach was followed in order to obtain an autonomous autorotation controller for the OH-58A helicopter. The main objective of the thesis was to show that reinforcement learning algorithms can be used to train machine agents without prior knowledge of a model or the environment, for the challenging task of safe landing through helicopter autorotation. Another objective of the study was to show that a reduction to the Height-Velocity restriction curve can be achieved by a machine agent, such that the machine agent can successfully land a helicopter, starting autorotation within the aforementioned H-V curve.

A point-mass model of the OH-58A helicopter in autorotation was built to be used in the reinforcement learning framework as the environment. The point-mass model is a variation of the widely used two-dimensional model introduced in Reference [14]. This mathematical model captures the governing dynamics of the helicopter autorotation, with two degrees of freedom in vertical plane and rotor dynamics. Since reinforcement learning tasks for complex dynamical system control problems are computationally expensive and time consuming, a simple model which captures the essential dynamics of autorotation was necessary. The point-mass model satisfies this necessity.

Reinforcement learning tasks need machine agents to infer actions from encountered states. Goal of the training algorithms of such machine agents is to obtain a mapping from states to actions that satisfies designer's specifications of how a task should be done. Such an agent was constructed by two neural network function approximators, namely the critic network and the actor network. The critic network's purpose is to learn and evaluate the value of each state, and then send this feedback to the actor. The actor network takes this feedback, and determines whether the agent's recent actions according to a policy led the agent to a better or worse state than the critic expected and update its parameters accordingly. The critic network learns the value of states by utilizing discounted returns from each training episode and updating towards minimizing a loss function to better represent each state's value. These returns are obtained by evaluating a reward function which is defined by designer to shape the desired behavior of the agent. The actor network's outputs were defined as center points of Gaussian distributions with standard deviations of 1. In order to facilitate learning, a machine agent needs exploration of different parts of the state and action spaces. The agent's policy is to sample from these distributions to determine actions at each state during training, in search of a better action which yields more rewards in the long run. Updates to the critic and the actor networks were found by a gradient based optimization algorithm called RMSProp. Gradients necessary for calculating the updates were found by backpropagation method.

After training was completed, performance of the agent was tested against various autorotation initial conditions both inside and outside the Height-Velocity diagram of OH-58A helicopter. During these tests, behavior of the agent was found to be similar to human pilots. The reinforcement learning agent achieves success criteria of $u < 3$ m/s and $w < 1$ m/s at the instant of touchdown for five out of six initial conditions. When the trajectories of these conditions were examined, it was observed that the agent had learned to execute a flaring maneuver for all starting conditions without any prior knowledge of such a maneuver's necessity or any guidance on how or when to execute a flare. In addition to this, the agent tries to achieve a sink rate near the minimum sink rate at approximately 23 m/s forward speed during the steady descent phase of the autorotation maneuver without such prior information during or after training. Autorotation trajectory with an entry height of 40 m and entry speed of 20 m/s is a clear evidence of this behavior, as the agent tilts the rotor disk forward following the entry and increases its forward speed slightly.

It was observed that rotor angular speed is maintained in the range as defined by the corresponding reward function term. However, it is evident that these limits can be tightened for better rotor speed control over the course of autorotation, since the agent clearly has the freedom in all test cases to further reduce the collective input just after autorotation entry to further arrest the decay in rotor angular speed. In addition to this, it was found that rotor loading limit was not violated during the test cases and the rotor stayed clear of the stall region.

Monte Carlo simulations of random autorotation entries at different heights and forward velocities for different gross weights were conducted. As it was shown, majority of the Height-Velocity restriction zone were cleared by the agent. Inside the knee of the H-V diagram the agent fails to achieve successful landing criteria for entry speeds up to 20 m/s and entry heights up to roughly 50 meters for 1360 kg gross weight. However, outside of this particular zone, the agent achieves successful landing except entry heights below 5 meters. It was shown that the area cleared by the agent for lower gross weight parameters increases with decreasing gross weight, since lower gross weight gives room for better energy management as there is more rotational energy of the rotor to be used for maneuvering instead of arresting the sink-rate. Touchdown speeds with respect to different entry conditions for different weights were also given. It was observed that touchdown speeds decrease with decreasing gross weights, which is also due to the fact that rotor kinetic energy left before touchdown is larger for low gross weight configurations. The agent uses this excess energy for a more cushioned touchdown.

Although the trained agent performed well, reinforcement learning methods such as the one used in this thesis have their own shortcomings. First, there is not a proof of guaranteed stability for nonlinear reinforcement learning controllers in the sense of control theory. Furthermore, the agent needs to extensively explore the state and action spaces to have a meaningful understanding of the problem at hand. This makes having a cluster of high performance computing machines a necessity for larger problems. Furthermore, real world application of such a reinforcement learning algorithm is not feasible for training in real time, due to the intensive processing and memory requirements of such algorithms. In addition, need for exploration inhibits real time training of safety critical equipment such as aircraft. However, trained agents can be deployed on aircraft, after extensive testing and high-fidelity simulation. Finally, reinforcement learning algorithms generally involve extensive tuning of parameters by trial-and-error.

Assuming that a feasible agent was trained for autonomous autorotation as described in this thesis, application of such an agent on a helicopter would need sensor data to operate. At its current state, the agent needs inertial forward and downward velocities, altitude above ground, and rotor angular speed. Velocities can be measured by an inertial navigation system, while altitude above ground information can be obtained by a radar altimeter. Rotor angular speed can be obtained by utilizing a shaft encoder or tachometer. Given all these sensor information, the agent can run on a flight computer continuously during autorotation for safe landing.

6.1 Future Work

In light of the conclusions derived above following items can be considered for future research in this thesis' subject:

- Actuator models for control inputs can be added for more realistic responses of the helicopter. This might necessitate changes to the agent architecture, such as using recurrent neural network function approximators instead of feedforward neural networks for actor and critic, in order to take time delay of the actuator models into account.
- Training can be conducted with a full 6-degree-of-freedom high-fidelity helicopter mathematical model for a more comprehensive analysis of reinforcement learning algorithms.
- Reinforcement learning can be applied on top of an automatic flight control system as a reference generator for upper modes of such a control system. This may help reduce the concerns on stability of the aircraft throughout flight.
- Trained agent can be integrated into a flight simulator and be used as a pilot training tool for autorotation.
- Stability of the dynamical systems under control by reinforcement learning agents can be further investigated for guarantee of stability proofs such as stability in the sense of Lyapunov.

REFERENCES

[1] P. Abbeel, A. Coates, T. Hunter, and A. Y. Ng. Autonomous autorotation of an rc helicopter. In *Experimental Robotics*, pages 385–394. Springer, 2009.

[2] C. W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37, 1989.

[3] E. N. Bachelder and B. L. Aponso. Using optimal control for rotorcraft autorotation training. In *Proceedings of the American Helicopter Society 59th Annual Forum, Phoenix, Arizona*. AHS International, 2003.

[4] P. Bibik and J. Narkiewicz. Helicopter optimal control after power failure using comprehensive dynamic model. *Journal of Guidance, Control and Dynamics*, 35(4):1354–1362, 2012.

[5] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[6] R. T. Chen and Y. Zhao. Optimal trajectories for the helicopter in one-engine-inoperative terminal-area operations. Technical report, National Aeronautics and Space Administration, Ames Research Center, 1996.

[7] R. Coulom. *Reinforcement learning using neural networks, with applications to motor control*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2002.

[8] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In *Advances in neural information processing systems*, pages 1017–1023, 1996.

[9] R. H. Crites and A. G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine learning*, 33(2-3):235–262, 1998.

[10] K. Dalamagkidis, K. P. Valavanis, and L. A. Piegl. Nonlinear model predictive control with neural network optimization for autonomous autorotation of small unmanned helicopters. *IEEE Transactions on Control Systems Technology*, 19(4):818–831, 2011.

[11] L. W. Dooley and R. D. Yeary. Flight Test Evaluation of the High Inertia Rotor System. Technical report, U.S. Army Research and Technology Laboratories (AVRADCOM), Forth Worth, Texas, 1979.

[12] A. Gessow and G. Myers. *Aerodynamics of the helicopter*. F. Ungar Pub. Co., 1952.

[13] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[14] W. Johnson. Helicopter Optimal Descent and Landing After Power Loss. Technical Report NASA-TM-73244, NASA Ames Research Center, 1977.

[15] W. Johnson. *Rotorcraft Aeromechanics*. Cambridge University Press, 2013.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] A. Y. Lee. *Optimal Landing Of A Helicopter In Autorotation*. PhD thesis, Stanford University, 1985.

[18] A. Y. Lee, A. E. Bryson, Jr., and W. S. Hindson. Optimal landing of a helicopter in autorotation. *13th Atmospheric Flight Mechanics Conference*, 11(1), 1986.

[19] D. J. Lee and H. Bang. Autonomous autorotation of an unmanned helicopter using a reinforcement learning algorithm. *Journal of Aerospace Information Systems*, 10(2):98–104, 2013.

[20] L.-J. Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[21] W. Meng and R. Chen. Study of helicopter autorotation landing following engine failure based on a six-degree-of-freedom rigid-body dynamic model. *Chinese Journal of Aeronautics*, 26(6):1380–1388, 2013.

[22] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[23] W. A. Pleasants and G. T. White. Status of improved autorotative landing research. *Journal of the American Helicopter Society*, 28(1):16–23, 1983.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[25] Z. Sunberg and J. Rogers. A fuzzy logic-based controller for helicopter autorotation. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, page 1150, 2013.

[26] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[27] S. Taamallah. A qualitative introduction to the vortex-ring-state, autorotation, and optimal autorotation. Technical report, National Aerospace Laboratory NLR, 2010.

[28] G. Tesauro. Practical issues in temporal difference learning. In *Advances in neural information processing systems*, pages 259–266, 1992.

[29] G. Tesauro. Td-gammon: A self-teaching backgammon program. In *Applications of Neural Networks*, pages 267–285. Springer, 1995.

[30] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[31] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[32] S. Tierney. *Autorotation path planning using backwards reachable set and optimal control*. PhD thesis, Penn State University, 2010.

[33] C. F. Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22(3-4):251–281, 1997.

[34] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[35] P. Werbos. Beyond regression: new tools for prediction and analysis in the behavioral sciences. *PhD thesis, Harvard University*, 1974.

[36] P. Werbos. Backpropagation: Past and future. In *Proceedings of the 2nd International Conference on Neural Network*. IEEE, 1988.

[37] G. White, A. Logan, and J. Graves. An evaluation of helicopter autorotation assist concepts. In *American Helicopter Society, Annual Forum, 38 th, Anaheim, CA*, pages 194–216, 1982.

[38] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[39] T. Wood. High energy rotor system. In *32th Annual V/STOL Forum of the American Helicopter Society*, 1976.

[40] T. Yomchinda, J. Horn, and J. Langelaan. Flight path planning for descent-phase helicopter autorotation. In *AIAA Guidance, Navigation, and Control Conference*, page 6601, 2011.

[41] T. Yomchinda, J. F. Horn, and J. W. Langelaan. Autonomous control and path planning for autorotation of unmanned helicopters. In *Proceedings of the American Helicopter Society 68th Annual Forum, Fort*. Citeseer, 2011.

APPENDIX A

ASYNCHRONOUS LEARNING ALGORITHM

Algorithm 1 Asynchronous Learning Algorithm

```
1: Initialize the global environment, actor, critic
2: Initialize global training episode counter $n = 1$
3: **repeat**
4:     Reset gradients $d\theta^q = 0$ and $d\theta^v = 0$
5:     Create worker with its own copy of environment, actor, critic
6:     Copy global parameters $\theta^q$ and $\theta^v$ to local network parameters $\theta^{q\prime}$ and $\theta^{v\prime}$
7:     $t_{start} = 1, t = t_{start}$
8:     **repeat**
9:         Get state $s_t$
10:        Perform action $a_t$ sampled from policy $\pi(s_t;\theta^{q\prime})$ with mean $\mu(s_t;\theta^{q\prime})$
11:        Get reward $r_t$ and next state $s_{t+1}$
12:        $t \leftarrow t+1$
13:    **until** $s_t$ is the terminal state
14:    $R = 0$
15:    **for** $i = t-1$ to $t_{start}$ **do**
16:        $R \leftarrow r_i + \gamma R$
17:        $d\theta^v \leftarrow d\theta^v - \frac{1}{2}\frac{\partial(R-V(s_i;\theta^{v\prime}))^2}{\partial\theta^{v\prime}}$
18:        $d\theta^q \leftarrow d\theta^q - (R - V(s_i;\theta^{v\prime}))\frac{1}{2}\frac{\partial(a_i-\mu(s_i;\theta^{q\prime}))^2}{\partial\theta^{q\prime}}$
19:    **end for**
20:    Perform global network parameter updates using $d\theta^v$ and $d\theta^q$ with RMSProp
       optimization
21:    $n \leftarrow n+1$
22: **until** $n > n_{max}$
```