

A COMPARATIVE STUDY OF DEEP REINFORCEMENT LEARNING
METHODS AND CONVENTIONAL CONTROLLERS FOR AERIAL
MANIPULATION

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

BY

KAZIM BURAK ÜNAL

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

FEBRUARY 2021

Approval of the thesis:

**A COMPARATIVE STUDY OF DEEP REINFORCEMENT LEARNING
METHODS AND CONVENTIONAL CONTROLLERS FOR AERIAL
MANIPULATION**

submitted by **KAZIM BURAK ÜNAL** in partial fulfillment of the requirements for
the degree of **Master of Science in Computer Engineering Department, Middle
East Technical University** by,

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

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Sinan Kalkan
Supervisor, **Computer Engineering, METU** _____

Prof. Dr. Afşar Saranlı
Co-supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Erol Şahin
Computer Engineering, METU _____

Assoc. Prof. Dr. Sinan Kalkan
Computer Engineering, METU _____

Assoc. Prof. Dr. Mehmet Serdar Güzel
Computer Engineering, Ankara University _____

Date:



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Kazım Burak Ünal

Signature :

ABSTRACT

A COMPARATIVE STUDY OF DEEP REINFORCEMENT LEARNING METHODS AND CONVENTIONAL CONTROLLERS FOR AERIAL MANIPULATION

Ünal, Kazım Burak

M.S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Sinan Kalkan

Co-Supervisor: Prof. Dr. Afşar Saranlı

February 2021, 67 pages

Aerial manipulation with unmanned aerial vehicles is increasingly becoming a necessity in many applications. In this thesis, we analyze the controller approaches for a bi-rotor aerial manipulator for a pick and place operation. First of all, we compare a classical control approach with a minimum snap trajectory generation and Deep Reinforcement actor-critic algorithms for the control of the aerial manipulator. Furthermore, we examine the effects of degrees of freedom of the manipulator for the Deep Reinforcement Learning approaches and analyze how the change of goal position of the object that the aerial manipulator needs to carry affects the training of the learning approaches. Moreover, to obtain a faster convergence for the learning approaches we have added informative states in which the aerial manipulator starts with the object it needs to carry is already grasped. Our results of the 2D simulation environment for the aerial manipulation suggest that all of the actor-critic algorithms yield valuable results with off-policy algorithms being more sample efficient. Still, these algorithms have stability issues that fail in the task for some cases. On the other hand, the classical controller approach does not have this problem but finishes the

task slower than the Deep Reinforcement Learning approaches.

Keywords: Aerial Manipulation, Deep Reinforcement Learning, Classical Control



ÖZ

DERİN PEKİŞTİRMELİ ÖĞRENME YÖNTEMLERİ VE GELENEKSEL KONTROLCÜLERİN HAVASAL MANİPULASYON İÇİN KARŞILAŞTIRMALI ÇALIŞMASI

Ünal, Kazım Burak

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Sinan Kalkan

Ortak Tez Yöneticisi: Prof. Dr. Afşar Saranlı

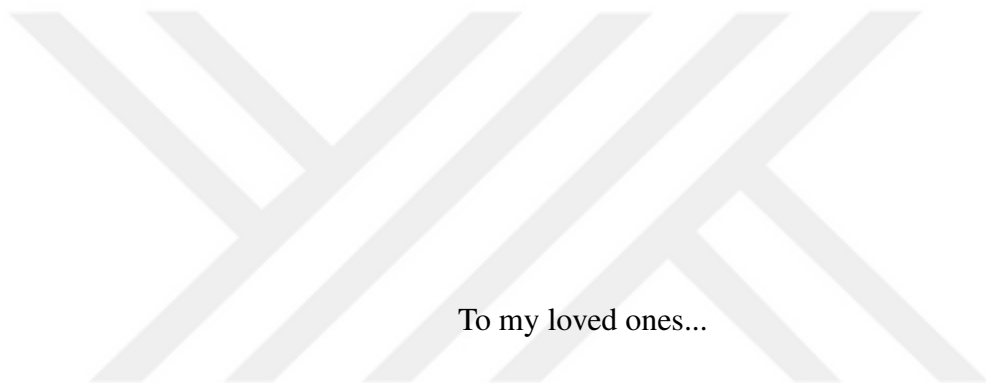
Şubat 2021 , 67 sayfa

İnsansız hava araçlarıyla havadan manipülasyon, birçok uygulamada giderek bir zorunluluk haline geliyor. Bu tezde, bir alma ve yerleştirme işlemi için kullanılan, çift rotorlu bir hava manipülatörü için kontrolcü yaklaşımlarını analiz ediyoruz. Hava manipülatörünün klasik bir kontrolcü ile minimum anlık yörünge oluşturma algoritmasını, Derin Güçlendirme aktör-kritik algoritmalarıyla karşılaştırıyoruz. Ayrıca, Derin Pekıştirmeli Öğrenme yaklaşımları için manipülatörün serbestlik derecelerinin etkilerini inceliyor ve hava manipülatörünün taşıması gereken nesnenin hedef konumunun değişiminin öğrenme yaklaşımlarının eğitimini nasıl etkilediğini analiz ediyoruz. Dahası, öğrenme yaklaşımları için daha hızlı bir yakınsama elde etmek için, hava manipülatörünün taşıması gereken nesne ile başladığı bilgilendirici durumlar ekledik. Hava manipülasyonu için 2D simülasyon ortamına ilişkin sonuçlarımız, tüm aktör-eleştirmen algoritmalarının değerli sonuçlar verdiğini ama politika dışı algoritmaların daha örnek verimli olduğunu gösterdi. Ancak bu algoritmaların hala bazı durumlarda görevin başarısızlığına neden olan kararlılık sorunları vardır. Öte yandan klasik kont-

rolc yaklařımında bu sorun yoktur, ancak Derin Pekiřtirmeli ğrenme yaklařımlarına gre grevi daha yavař bitirir.

Anahtar Kelimeler: Havada Manipulasyon, Derin Pekiřtirmeli ğrenme, Klasik Kontrol





To my loved ones...

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my supervisors Assoc. Prof. Dr. Sinan Kalkan and Prof. Dr. Afşar Saranlı for their support and guidance during my thesis study. I would also like to thank the examining committee members; Assoc. Prof. Dr. Erol Şahin and Assoc. Prof. Dr. Mehmet Serdar Güzel for their comments and critics.

I would like to express my gratitude to ASELSAN, for their support for my thesis study.

I would also like to thank my colleagues Sinan Pakkan, Ali Erkin Arslan, Burhan Ölmez and Cantürk Sanan for their encouragement during this journey.

Finally, I owe my loving thanks to my family, my loving mother Cemran, my father Ahmet, my brother Berk and my cousin Berke for their undying love, support and encouragement.

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vii
ACKNOWLEDGMENTS	x
TABLE OF CONTENTS	xi
LIST OF TABLES	xiv
LIST OF FIGURES	xv
LIST OF ABBREVIATIONS	xviii
LIST OF VARIABLES	xix

CHAPTERS

1 INTRODUCTION	1
1.1 Motivation and Problem Definition	1
1.2 Contributions	3
1.3 The Outline of the Thesis	4
2 RELATED WORK	5
2.1 Classical Control Approaches	5
2.2 Learning-Based Approaches	7
2.2.1 Manipulation	8

2.2.2	Unmanned Aerial Vehicle Control	9
2.2.3	Mobile Base Manipulation	9
2.3	Summary	10
3	MATHEMATICAL BACKGROUND	11
3.1	Dynamic Model of Aerial Manipulator	11
3.1.1	Equations of Motion	12
3.2	Aerial Manipulator as a Markov Decision Process	15
3.3	Reinforcement Learning	17
3.4	Deep Reinforcement Learning	21
3.4.1	Value Gradient Methods	21
3.4.2	Policy Gradient Methods	22
4	METHODS	25
4.1	Classical Control Approach	25
4.1.1	Trajectory Generation	25
4.1.2	Conventional Controller Design	26
4.2	Learning-based Approaches	28
4.2.1	On-policy Algorithms	29
4.2.1.1	Trust Region Policy Optimization	29
4.2.1.2	Asynchronous Advantage Actor Critic	30
4.2.1.3	Proximal Policy Optimization	31
4.2.2	Off Policy Algorithms	31
4.2.2.1	Deep Deterministic Policy Gradient	32
4.2.2.2	Twin Delayed Deep Deterministic Policy Gradient	32

4.2.2.3	Soft Actor Critic	33
5	SIMULATION ENVIRONMENT AND SIMULATION DETAILS	35
5.1	Design of the Simulation Environment	35
5.2	States and Actions of the Simulation	37
5.3	Initial State Constraints	38
5.4	Termination Criteria	38
5.5	Reward Shaping	39
6	EXPERIMENTS AND RESULTS	41
6.1	Implementation and Optimization Details	41
6.1.1	Optimization of Classical Controller Approach	41
6.1.2	Deep Reinforcement Learning Approaches Implementations	44
6.2	Analyzing Learning-based Controllers	44
6.2.1	Analyzing On-policy Algorithms	45
6.2.2	Analyzing Off-policy Algorithms	46
6.2.3	On-policy vs. Off-policy Algorithms	48
6.3	Learning-based vs. Conventional Control Comparison	49
6.4	Analyzing the Effects of Different Factors	50
6.4.1	Effects of Goal and Start Position on the Learning-based Approaches	50
6.4.2	Effects of DoF on the Learning-based Approaches	52
6.4.3	Effects of Instructive State on the Learning-based Approaches	54
7	CONCLUSION	57
	REFERENCES	61

LIST OF TABLES

TABLES

Table 5.1	Reward function parameters	39
Table 6.1	Optimized controller parameters	43
Table 6.2	Training computation results for on-policy and off-policy algorithms	50
Table 6.3	Best model results. Results for reinforcement learning algorithm are obtained by running 10 trial that each contain 500 random starting point	50

LIST OF FIGURES

FIGURES

Figure 1.1	The AMUSE Unmanned Aerial Manipulator [1].	2
Figure 1.2	A bald eagle hunting a prey in the water [2].	3
Figure 2.1	Comparison of eagle and quadrotor that performs avian like grasping through captions [2].	6
Figure 2.2	Planning and control approaches for UAMS [3].	6
Figure 2.3	Lego stacking task stages. From left to right shows the starting state, reaching, grasping and stacking [4].	8
Figure 2.4	Dual UR5 arm Husky robot for picking operation [5].	10
Figure 3.1	Reference frames	12
Figure 3.2	Forces and moments acting on the system	13
Figure 3.3	Overall view of the agent-environment interaction	18
Figure 3.4	DQN block diagram	23
Figure 3.5	Actor-Critic architecture high level block diagram.	24
Figure 4.1	Conventional controller block diagram.	26
Figure 4.2	Aerial Manipulator control diagram.	28

Figure 5.1	An instance of Aerial manipulation for pick and place operation. The green dot is the goal position of the box which is the blue squared shape object. The dark red line is the Aerial manipulator and the light red lines indicate rotor thrusts. The red cross is the aim of the left gripper end-point location and the light yellow cross is the goal position of the aerial manipulator. The yellow line under the blue box can be considered as ground.	36
Figure 6.1	Random Starting points for Controller Optimization	42
Figure 6.2	Desired and observed trajectory of positions (left) and manipulator arm angle (right). Blue line represents the observed position of the aerial vehicle and the brown line the trajectory generated by using the minimum snap trajectory algorithm.	43
Figure 6.3	Desired and observed trajectory of velocity in the x-direction (left) and y-direction(right). The blue line represents the observed velocity of the aerial vehicle and the brown line represents the trajectory generated by using the minimum snap trajectory algorithm.	44
Figure 6.4	Mean episode reward of on-policy algorithms during training. Dark lines represent the mean episode reward and the light regions shows the standard deviation.	45
Figure 6.5	Mean episode length (left) and mean episode success rate (right) of on-policy algorithms during training. Dark lines represent the mean episode reward and the light regions show the standard deviation. . . .	46
Figure 6.6	Mean episode reward of off-policy algorithms during training. Dark lines represent the mean episode reward and the light regions shows the standard deviation.	47
Figure 6.7	Mean episode length (left) and mean episode success rate (right) of off-policy algorithms during training. Dark lines represent the mean episode reward and the light regions shows the standard deviation. . . .	47

Figure 6.8	Off-policy (SAC) and on-policy (TRPO) comparison graph that shows the Mean episode reward (left) and mean success rates (right) . Dark lines represent mean whereas the light regions represents the standard deviation.	48
Figure 6.9	Off-policy (SAC) and on-policy (TRPO) mean reward comparison in terms of time steps.	49
Figure 6.10	Comparison of mean reward of the SAC algorithm for different destination positions of the box.	52
Figure 6.11	Point Distribution of the best models obtained from the algorithms (a) A2C, (b) TRPO, (c) PPO. Blue dots represents the successful starting points and the red dots represents the failed starting points. . .	53
Figure 6.12	Point Distribution of the best models obtained from the algorithms (a) DDPG, (b) TD3, (c) SAC. Blue dots represents the successful starting points and the red dots represents the failed starting points. . .	54
Figure 6.13	Comparison of mean reward (left) and mean length (right) of the SAC algorithm for different degree of freedom manipulators.	55
Figure 6.14	Comparison of mean reward of the SAC algorithm for adding informative state to the training.	55

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
UAV	Unmanned Aerial Vehicle
UAM	Unmanned Aerial Manipulator
PID	Proportional-Derivative-Integral
LQR	Linear Quadratic Regulator
DoF	Degree of Freedom
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
KL	Kullback-Leibler
CPU	Central Processing Unit
QP	Quadratic Programming
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization
A3C	Asynchronous Advantage Actor Critic
DDPG	Deep Deterministic Policy Gradient
TD3	Twin Delayed Deep Deterministic Policy Gradient
SAC	Soft Actor-Critic
DQN	Deep Q Learning
CoF	Center of Mass

LIST OF VARIABLES

v_{p_x}	Velocity of aerial manipulator in x-direction
v_{p_y}	Velocity of aerial manipulator in y-direction
β	Angle between left gripper and manipulator
γ	Angle between manipulator and aerial vehicle
θ	Orientation of aerial vehicle
ψ	Orientation of box object
p_{p_x}	Position of aerial manipulator COF without manipulator in x-direction
p_{p_y}	Position of aerial manipulator COF without manipulator in y-direction
p_{g_x}	Position of left gripper end point in x-direction
p_{g_y}	Position of left gripper end point in y-direction
p_{b_x}	Position of box object in x-direction
p_{b_y}	Position of box object in y-direction
M	Mass of the aerial vehicle
m	Mass of the box shaped object
I_p	Inertia of aerial vehicle
I_m	Inertia of the manipulator
l	Length of the manipulator
d	Half of the length of aerial vehicle



CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Recent studies have shown that the usage of unmanned aerial vehicles (UAVs) is rapidly increasing not only in the research community but also in the industry and in daily life [6]. Common applications of UAVs include surveillance [7], search and rescue [8], transportation of slung loads [9] and inspection of transmission lines [10]. Though these accomplishments solved some of the problems of the industry, these functions are mostly about "seeing" the world. In order to increase the utilization of UAVs, in recent years, researchers have started to explore applications where UAVs are needed to interact with the world physically with the help of a manipulator. This is generally known as aerial manipulation.

Aerial manipulation tasks consist of grasping, perching, transporting, and (re-)positioning of some objects with UAVs. UAVs that are used for aerial manipulation tasks are called Unmanned Aerial Manipulators (UAMs), which usually consist of two parts; the UAV itself and a mechanism like a robotic arm that enables the UAV to interact with the world, which improves the capabilities of the UAV. An example of UAM is shown in Figure 1.1.

Aerial manipulation can be considered as a subset of mobile base manipulation with some noticeable differences from ground mobile manipulators. Some of the differences are:

- Due to not having a stable base, the actions done by the robotic arm directly affect the UAV's pose and stability.

- The UAV's dynamic changes when the system gets close to the ground or a wall due to the airflow changes, which is called the ground effect. The ground effect makes the system harder to control.
- Since the UAVs, especially quad-rotor UAVs, have non-linear coupled dynamics, they are harder to control.
- Finally, as the payload capacity of the UAVs are limited, they are unable to house dexterous robotic manipulators.



Figure 1.1: The AMUSE Unmanned Aerial Manipulator [1].

In addition to these challenges, due to the battery limit, there is also the time constraint to be considered. According to Thomas et al. [2] one of the limitations of the initial approaches to the grasping problem is the need to reach the target object with a near-zero approach velocity which results in unnecessary energy consumption since maintaining a significant speed of the UAV, while manipulation task is being performed, reduces the required flight time which results in less energy consumption. This high-speed manipulation task, also observed in birds (see Figure 1.2), can be called dynamic manipulation in robotics and it requires agile flight and manipulation

which makes the system more challenging than it is already.

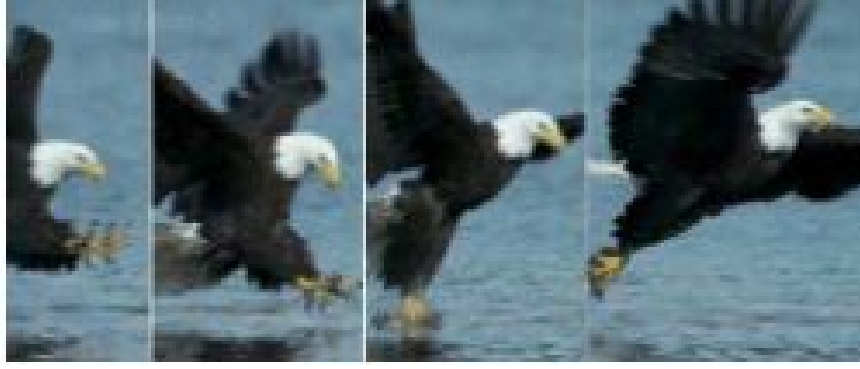


Figure 1.2: A bald eagle hunting a prey in the water [2].

In this thesis, in order to solve the task of a bi-rotor aerial manipulator that has to pick an object and move it to a given destination, we compare (i) a conventional controller with some trajectory generation algorithm that enables the dynamic grasping and (ii) various end-to-end deep reinforcement learning approaches that consist of some of the policy-gradient actor-critic methods. In this problem, the algorithms have to stabilize and control the UAV and the manipulator as well as plan a trajectory for grasping and transportation of the object to a target location.

1.2 Contributions

In this thesis, our contributions are as follows:

- We have created an environment that simulates aerial manipulation pick and place tasks. In this environment, the aerial manipulator starts from a point and has to grasp a square-shaped object and move it to a given destination. The algorithms that are required to solve the problem have to plan a trajectory for the grasping and transportation of the object while controlling the manipulator and the platform.
- We conducted a comparison between three learning-based off-policy actor-critic methods named, Deep Deterministic Policy Gradient (DDPG) [11], Twin Delayed Deep Deterministic Policy Gradient (TD3) [12], Soft Actor-Critic (SAC)

[13], and three learning-based on-policy actor-critic methods named Trust Region Policy Optimization (TRPO) [14], a deterministic synchronous version of Asynchronous Advantage Actor-Critic (A3C) method [15], Proximal Policy Optimization (PPO) [16] for the pick and place operation. In addition to this, we have also compared conventional controller [17] with minimum snap trajectory [18] and learning-based approaches on the same problem.

- Lastly, we have applied instructive states for the learning-based approaches to improve the training process of the learning-based approaches. Adding an instructive state for the training improves the learning process by reducing the exploration time for the problem of manipulators where information is usually obtained if and only if the goal is reached [4]. Based on this, for some episodes in the training, different from the normal training processes, we have started the episode in random starting points where the aerial manipulator already grasped the squared object so that the learning-based algorithms learn faster.

1.3 The Outline of the Thesis

This thesis outline is as follows. In Chapter 2, the related work proposing to solve the manipulation task of picking and placing an object in terms of control-based approaches, and learning-based approaches are presented. In Chapter 3, necessary mathematical backgrounds are given about the bi-rotor system dynamics, reinforcement learning, and deep reinforcement learning. The details of the methods implemented in this thesis are explained in Chapter 4 and the simulation environment details are given in Chapter 5. In Chapter 6, experimental results are shown. Lastly, in Chapter 7, the thesis is concluded with a discussion of future research directions.

CHAPTER 2

RELATED WORK

In this chapter related work about classical control approaches for the control of UAVs and UAMs is given in Section 2.1 and in Section 2.2 related work about learning-based approaches for the control problem of manipulators, UAVs, and mobile-based manipulator is given. Finally, in Section 2.3 we give a brief summary about deep reinforcement learning approaches for the UAMs control.

2.1 Classical Control Approaches

There is abundant literature about controlling UAVs using the classical control approach. A review about the control algorithms like PID (Proportional Integral Derivative), LQR (Linear-Quadratic Regulator), H_∞ and adaptive control is conducted that discusses the advantages and weaknesses of these algorithms for controlling the UAVs like quadrotors [19]. Moreover, comprehensive literature surveys about trajectory planning for UAVs are done [20, 21]. In addition to these, a survey that covers the manipulation and grasping of robotic arms is provided [22].

Though these papers give a comprehensive study about manipulators and UAV control, they do not cover UAM control and its challenges. Due to the presence of the manipulator and its interactions with the environment, the dynamics and the coupling effects of the system are very different and more complex than the UAVs. However, because of the addition of a manipulator, UAMs can solve much more complex tasks. The first research about aerial manipulators is mostly about UAVs with a fixed gripper attached to it [23, 24]. After these, more advanced tasks like valve turning and transportation of some load are covered [25, 26, 27]. Moreover, An avian-like grasp-

ing by a quadrotor with the use of a minimum snap trajectory for path planning and a non-linear controller for following this trajectory is presented [2]. The minimum snap trajectory enables the UAM to perch an object in high velocities that resembles the perching motion of eagles, as shown in Figure 2.1.



Figure 2.1: Comparison of eagle and quadrotor that performs avian like grasping through captions [2].

Recent studies about UAMs [3, 28] showed that the problem of designing a motion planning and control algorithm for the UAMs can be clustered into two approaches (see Figure 2.2). The first one is the centralized approach where the UAV and the manipulator are considered as one, so the algorithms are designed from one model. The second one is the decentralized approach where the UAV and the manipulator are considered as separate entities so the manipulator acts as a disturbance to the UAV and vice versa.

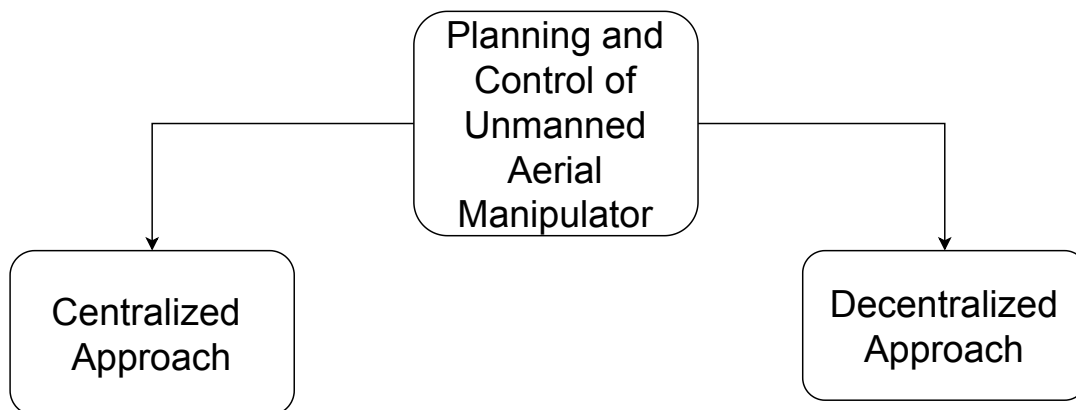


Figure 2.2: Planning and control approaches for UAMS [3].

The centralized approach can be further divided into two in terms of the modeling of the system. First, one being the standard Euler-Lagrangian representation of the system [29] which yields a representation of the dynamic model as a matrix, and the second one being the Newton-Euler formulation that is simpler to code [30, 31]. No matter which approach of representing the dynamics of the system is used, quite a number of model-based approaches like LQR based approach [32], adaptive sliding mode approach [33], Lyapunov-based adaptive control with gain-scheduling approach [34] have been developed for the centralized approach.

In the decentralized approach, since the UAV and the manipulator are considered as separate bodies, their control is also considered separately but required to have some robustness to overcome the disturbances that come from the other body. One of the examples of this approach where a multi-layer architecture is used for the control of a UAV that has a robot arm [35]. Moreover, a simple PID controller and a back-stepping controller are proposed [36]. In addition to these, a task-based control strategy for the UAV and the switching-base controller for the manipulator is designed [37] to have a simple controller for the UAMs.

2.2 Learning-Based Approaches

With the advances in deep learning methods, it is possible to make use of deep reinforcement learning approaches for the control of dynamic systems like manipulators, UAVs, and mobile-based manipulators. One of the first deep reinforcement learning (DRL) algorithms called Deep Q Networks (DQN) [38] has shown that it is possible to train a deep neural network to play some Atari games just like a human being by only observing the raw pixel data and by defining a simple reward that shows whether the algorithm won or lost the game.

In recent years, different approaches [39] than DQN have been proposed in the literature which can be gathered under the name "policy gradients". It has been shown that these algorithms are better in continuous domain problems. After these methods, to improve the convergence and the stability of the learning algorithms, "Actor-Critic Networks" [40, 11] that train two network architectures together have emerged. One

of the networks, called the actor-network, tries to estimate the best action at a given time and the other one, called the critic network, tries to estimate the value of each state in the domain. With these improvements and more, DRL has been used in some robotic problems that involve motion planning and control problems like manipulation [41] and UAV control [42].

2.2.1 Manipulation

Due to the complex nature of manipulators, learning-based control approaches are becoming more and more popular to solve the manipulation tasks of robotic arms for problems like a door opening [41], and pick and place operations [43, 4]. Popov et al. [4] have shown that it is possible to train a neural network to pick, move and place a Lego piece onto another one with a 9 DoF (Degree of Freedom) robotic arm which can be seen in Figure 2.3. They have used reward shaping where some domain knowledge is required to give some reward for every transition from one state to another that decreases the exploration time. Also, to increase the learning speed instead of always starting from 2 Lego pieces on the table, they have initialized the robotic arm where it holds one of the Lego pieces and called these special starting states "Instructive states" to reduce exploration even more.

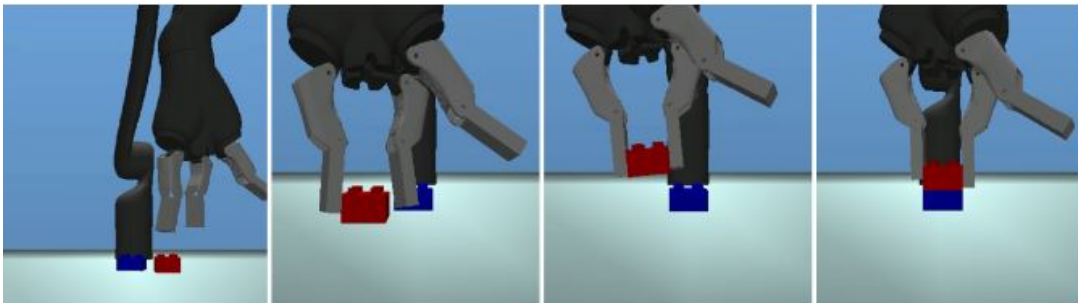


Figure 2.3: Lego stacking task stages. From left to right shows the starting state, reaching, grasping and stacking [4].

2.2.2 Unmanned Aerial Vehicle Control

Deep reinforcement learning has also been applied in the control problem of UAVs. It has been shown that it is possible to control attitude for UAVs with the PPO algorithm both in simulation and the real world [44, 45]. Moreover, Hwangbo et al. [46] used reinforcement learning for position control which directly maps UAV states to actuator commands. Furthermore, it has been illustrated one PPO algorithm can be used as a controller for multiple different sizes of UAVs without the need of re-training or re-tuning and it is possible to follow a trajectory while reaching a waypoint with deep reinforcement learning approaches [42, 47].



2.2.3 Mobile Base Manipulation

There is also research on using deep reinforcement learning for mobile manipulation. One of the examples for this is where a hierarchical reinforcement learning architecture, in which some of the sub-goals were masked to decide the phase of the agent to reduce the energy consumption and unnecessary motion, was proposed to solve a door opening task for the mobile manipulator [48]. In addition to this, a demonstration-based method was proposed [49] and a PPO-based algorithm was used for a mobile manipulator which was also tested in the real world [5], – see Figure 2.4. Also, PPO and DDPG based algorithms were compared where the mobile manipulator first has to avoid an obstacle and then has to reach an object [50].



Figure 2.4: Dual UR5 arm Husky robot for picking operation [5].

2.3 Summary

We observe that the literature has widely used deep reinforcement learning for controlling UAVs, manipulation, and mobile-based manipulation. However, to the best of our knowledge, there is no published work about UAMs for pick and place operation in terms of deep reinforcement learning approaches.

CHAPTER 3

MATHEMATICAL BACKGROUND

In this chapter, first, we briefly formalize the dynamic model of the aerial manipulator in Section 3.1. Then, to apply deep reinforcement learning to our system we convert it to a Markov Decision Process in Section 3.2. After that, we briefly explain the background of the deep learning algorithms that we use in this thesis in Sections 3.3 and 3.4.

3.1 Dynamic Model of Aerial Manipulator

This section formally represents the motion of the bi-rotor aerial manipulator where a bi-rotor is a planar version of a quad-rotor. These equations are obtained using the Euler-Lagrange equations which are more practical for seeing the coupled dynamics of multi-body vehicles such as an aerial manipulator [51]. In this thesis, we have separated the equations of motions into two sets: one for the UAV and the other one for the manipulator (see details [37]) to see the overall dynamic coupling of the system better. The UAV has a manipulator with a gripper, attached to it at the center of its mass with an actuated revolute joint.

As it can be seen in Figure 3.1, (e_x, e_y) is the inertial frame coordinates that have the origin point O^i and (e_1, e_2) is used for aerial body frame whose origin O^b is located at CoF (Center of Mass) of the aerial vehicle. Lastly, (e_b, e_a) is used for defining the manipulator body frame coordinate system whose origin O^m is also located at the CoF of the aerial vehicle.

Figure 3.2 depicts the clockwise rotation of the aerial vehicle under the counterclock-

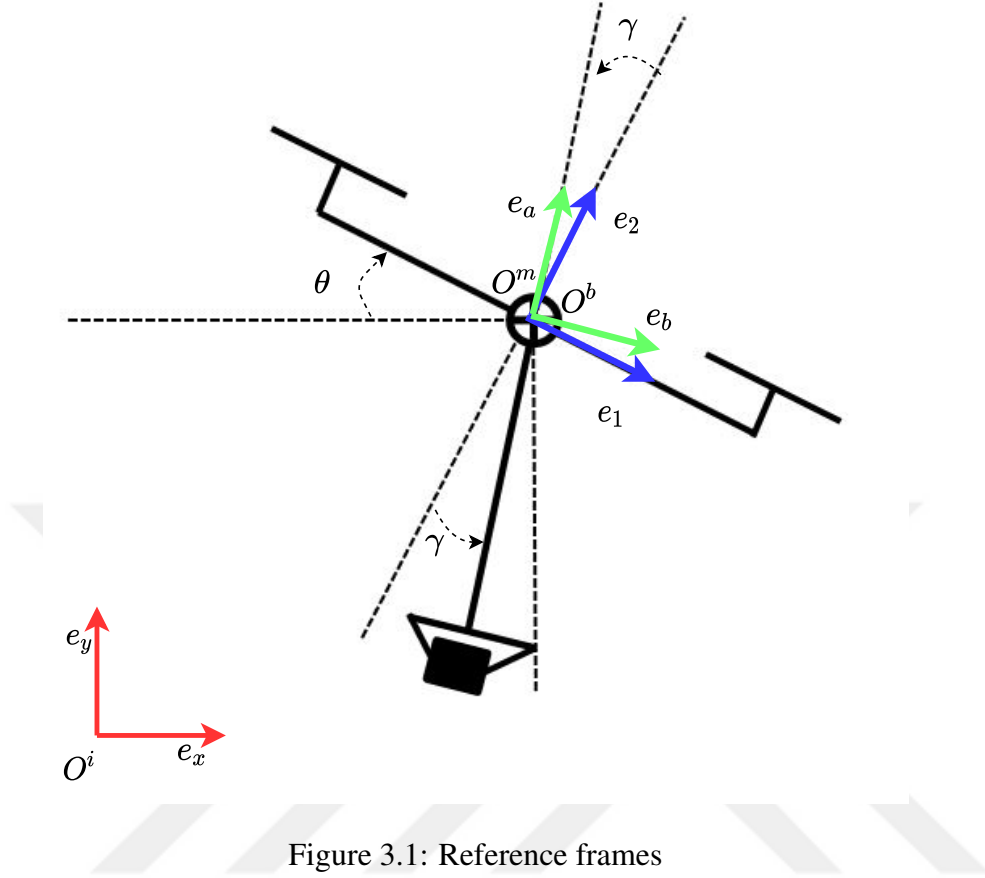


Figure 3.1: Reference frames

wise rotation of the manipulator and shows the forces and moments acting on the body. The rotational displacement shown in Figure 3.2 can be understood by the transformation matrices $R^\gamma \in \mathbb{R}^{2 \times 2}$, where γ is the orientation of the manipulator with respect to the aerial vehicle, and $R^\theta \in \mathbb{R}^{2 \times 2}$ where θ is the attitude of the aerial vehicle.

3.1.1 Equations of Motion

Aerial manipulator centralized equation of motion is obtained with the use of Euler-Lagrange formulation [37]. According to them, the translation can be modeled as:

$$f_x = (M + m)\dot{v}_x - ml \cos(\theta - \gamma)(\ddot{\theta} - \ddot{\gamma}) + ml \sin(\theta - \gamma)(\dot{\theta} - \dot{\gamma})^2, \quad (3.1)$$

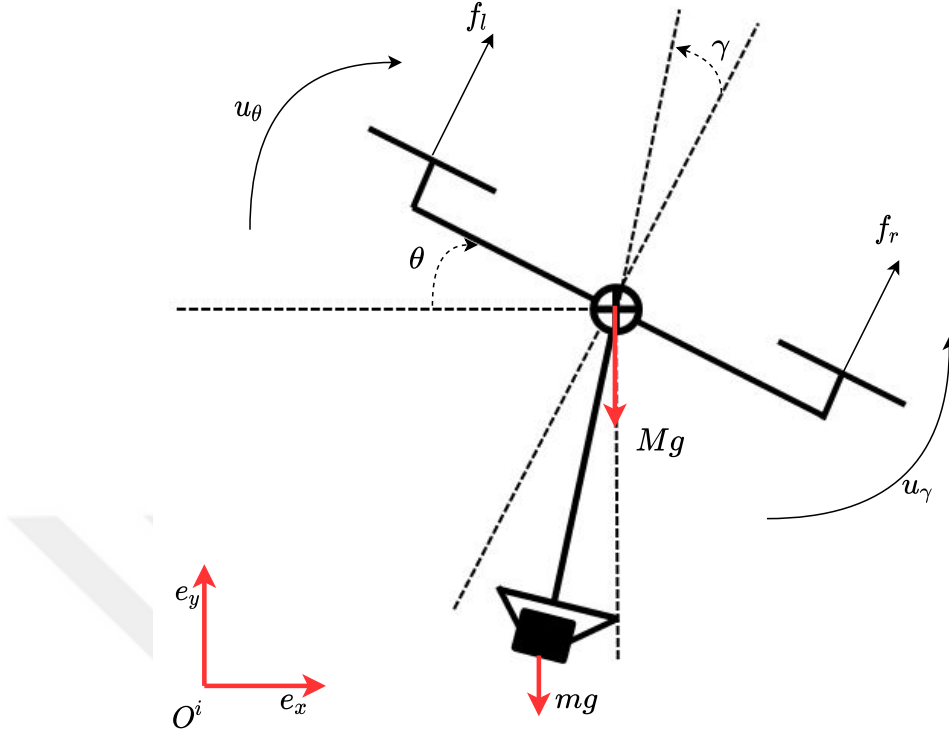


Figure 3.2: Forces and moments acting on the system

$$f_y = (M + m)\dot{v}_y + ml \sin(\theta - \gamma)(\ddot{\theta} - \ddot{\gamma}) + ml \cos(\theta - \gamma)(\dot{\theta} - \dot{\gamma})^2 + (M + m)g, \quad (3.2)$$

where f_x is the thrust in x-direction in inertial frame and f_y is the thrust in y-direction in inertial frame. On the other hand, the rotational transformations are:

$$u_\theta = (I_p + ml^2 + I_m)\ddot{\theta} - (ml^2 + I_m)\ddot{\gamma} - ml \cos(\theta - \gamma)\dot{v}_x + ml \sin(\theta - \gamma)\dot{v}_y + mgl \sin(\theta - \gamma), \quad (3.3)$$

$$u_\gamma = (ml^2 + I_m)\ddot{\gamma} - (ml^2 + I_m)\ddot{\theta} + ml \cos(\theta - \gamma)\dot{v}_x - ml \sin(\theta - \gamma)\dot{v}_y - mgl \sin(\theta - \gamma), \quad (3.4)$$

and from these, the decentralized equations are obtained for the aerial vehicle only as given by Escareno et al. [37]. Translation motion for the aerial vehicle is mathematically defined as:

$$(M + m)\ddot{\eta} = \mathbf{F}_\eta + \mathbf{W} + \mathbf{F}_c + \mathbf{F}_t, \quad (3.5)$$

where η represent the 2D position vector defined as ,

$$\eta = \begin{bmatrix} x \\ y \end{bmatrix}, \quad (3.6)$$

and W is the total weight vector given as,

$$W = \begin{bmatrix} 0 \\ -(M + m)g \end{bmatrix}. \quad (3.7)$$

Also, F_c and F_t corresponds to the centrifugal and tangential forces respectively, that occur due to the manipulator. They are defined as

$$F_c = \begin{bmatrix} ml \sin(\theta - \gamma)(\dot{\theta} - \dot{\gamma})^2 \\ ml \cos(\theta - \gamma)(\dot{\theta} - \dot{\gamma})^2 \end{bmatrix}, \quad (3.8)$$

$$F_t = \begin{bmatrix} ml \cos(\theta - \gamma)(\ddot{\theta} - \ddot{\gamma}) \\ -ml \sin(\theta - \gamma)(\ddot{\theta} - \ddot{\gamma}) \end{bmatrix}. \quad (3.9)$$

Finally F_η is the control input thrust vector and is defined as

$$F_\eta = R^\theta F^b = \begin{bmatrix} (f_l + f_r) \sin \theta \\ (f_l + f_r) \cos \theta \end{bmatrix}. \quad (3.10)$$

On the other hand, rotational motion for the aerial vehicle is defined as,

$$I_p \ddot{\theta} = \tau_\theta + \tau_{\theta_t} + \tau_{\theta_w} + \tau_{\theta_\eta}, \quad (3.11)$$

where the disturbing torques created by the translational motion of the manipulator mass τ_{θ_η} is defined as,

$$\tau_{\theta_\eta} = ml \cos(\theta - \gamma) \dot{v}_x - ml \sin(\theta - \gamma) \dot{v}_y, \quad (3.12)$$

and the disturbance of manipulator and mass is defined as

$$\tau_{\theta_w} = -mgl \sin(\theta - \gamma), \quad (3.13)$$

$$\tau_{\theta_t} = (ml^2 - I_m)(\ddot{\theta} - \ddot{\gamma}). \quad (3.14)$$

Finally, τ_θ is the control input defines as,

$$\tau_\theta = (f_r - f_l)d. \quad (3.15)$$

In addition to the translational and rotational dynamics of the aerial vehicle, the dynamics of the manipulator is also given by Escareno et al. [37] as,

$$(I_m + ml^2)(\ddot{\theta} - \ddot{\gamma}) = \tau_\gamma + \tau_{\gamma_w} + \tau_{\gamma_\eta}, \quad (3.16)$$

where the disturbance of the gripped object τ_{γ_w} and the disturbance of the aerial vehicle τ_{γ_η} are defined as

$$\tau_{\gamma_w} = mgl \sin(\theta - \gamma), \quad (3.17)$$

$$\tau_{\gamma_\eta} = -ml \cos(\theta - \gamma)\dot{v}_x + ml \sin(\theta - \gamma)\dot{v}_y. \quad (3.18)$$

3.2 Aerial Manipulator as a Markov Decision Process

In this section, we first provide the basics of a Markov Decision Process and then present the aerial manipulator control problem as a finite horizon MDP (Markov Decision Process) since according to Sutton and Barto [52], MDP is the mathematical foundation of a reinforcement learning solution for a given control problem.

MDPs are the mathematical skeletons of a sequential decision-making process, in which every action not only affects its next states but also the subsequent states. So with MDPs, some theoretical declarations can be made for the reinforcement learning problem. In an MDP, the learning algorithm, which is also a decision-maker, is called the *agent*. Everything that does not include the agent and all of the things that the agent interacts with is called the *environment*. These two always interact with each other in such a way that the agent chooses an action and the environment responds to the agent by producing new states with the addition of some reward. The reward is a numerical value, which is what the agent tries to maximize over the whole interaction with the choice of actions.

An MDP can be represented with a tuple (S, A, P, R, ρ_0) where S denotes the set of all states in the process and A the set of possible action of the process. Moreover,

MDP describes a sequence of discrete time steps $t = 0, 1, 2, 3, 4, 5, \dots$ such that the process is at state $s \in S$ and the agent takes an action $a \in A$ according to this state s at time step t . As a result of this action, one step later, the environment returns to the agent with a numerical value called reward $r_{t+1} \in R$ with the new state s_{t+1} . Therefore, R represents the set of possible reward values of the process. If the MDP has a finite horizon, which is the case in our problem, S , A and R have finite number of elements. Then, the transition from s to s_{t+1} when the action a has been taken and the reward r_{t+1} is received, has a well-defined discrete probability distribution given as P , which only depends on the current state and action, and is true for all states, actions and rewards in the process:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, s_2, s_3, \dots, s_t), \quad (3.19)$$

which is called the *Markov Property*, which states that subsequent states and rewards only depend on the current states and actions and not before.

By following Equation 3.19, a trajectory can be generated by starting from an initial state s_0 that has a probability distribution such as:

$$\rho_0(s) = Pr\{s_0 = s\}, \quad (3.20)$$

and the trajectory can be seen as:

$$s_0, a_0, s_1, r_1, a_1, s_2, r_2, a_2, s_3, r_3, \dots \quad (3.21)$$

During this trajectory, the agent, which is the decision maker, selects the actions by following a function that is also called *policy* which can be given as a probability distribution over actions given the current state:

$$\pi(a|s) = Pr\{a_t = a|s_t = s\}. \quad (3.22)$$

With these and the dynamic model of the aerial manipulator, it is possible to represent the system as a Markov Decision Process such that the states of the aerial manipulator can be defined as:

$$s_t = [p_{bx}, p_{by}, p_{px}, p_{py}, p_{gx}, p_{gy}, \psi, \theta, \gamma, \beta, v_{px}, v_{py}, \dot{\theta}], \quad (3.23)$$

and the actions of the aerial manipulator can be defined as:

$$[f_l, f_r, v_{j_{arm}}, v_{j_{grripper}}]. \quad (3.24)$$

Since we know the dynamic model of the system, the MDP state-transitions can be re-defined with discrete-time. In other words, the actions u that are explained in Section 3.1 in a small-time step Δt are constant and the resulting state of the MDP is the state of the aerial manipulator at that time step. Note that this formalization still satisfies the Markov Property, since the dynamic equations of motion of the aerial manipulator still just depend on the current state and not the previous states. In addition to this, by using a difference equation that can be close enough to the differential solution if a small-time step is given, Equations shown in Section 3.1 can be re-written in a discrete-time domain instead of a continuous one [17].

With these, since the underlying dynamics are known, there is no stochasticity in the problem so the Equation 3.19 can be re-defined in a deterministic way as:

$$s_{t+1} = P(s, a). \quad (3.25)$$

The definitions of the reward signal and the initial state distribution of the aerial manipulator are given in Chapter 4.

3.3 Reinforcement Learning

Reinforcement learning is a Markov Decision Process where learning occurs with the help of the reward as the supervisory signal [52]. Reinforcement learning is a closed-loop problem where actions were taken by the agent to influence the state which is the input of the agent. In Reinforcement learning, the agent is not told which action to take during the learning process and is required to *learn* which action to take that yields the maximum reward by, in most simple terms, trying. The maximum return can either be the immediate reward that the agents receive or the cumulative future rewards, which is mostly the case. The states that the environment yields to the agent are affected by the actions that the agent has chosen and the dynamics of the environment in a period. So to summarize, in the reinforcement learning problem, the agent tries to learn the best actions that will result in maximum reward by using the interaction between itself and the environment which can be seen in Figure 3.3.

Since reinforcement learning can be formalized as a Markov Decision Process, the agent uses a policy that takes states as input to output actions to be taken, and as a

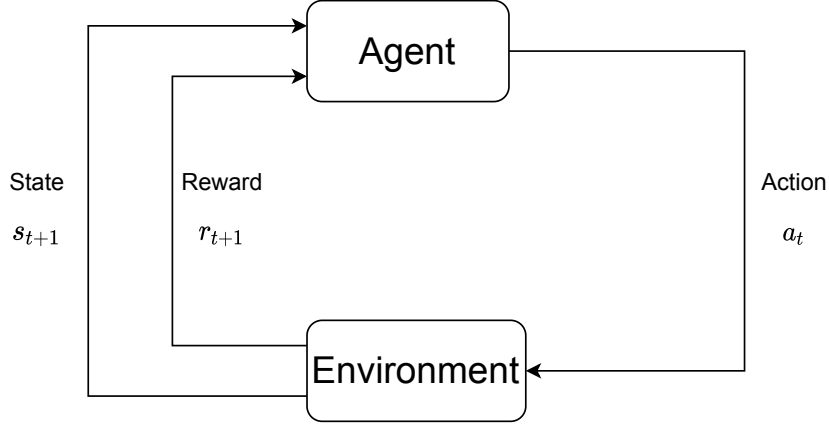


Figure 3.3: Overall view of the agent-environment interaction

result of this, the environment yields the new states and the reward to the agent with the actions that the agent has taken and according to the dynamics of the environment. This describes the reinforcement learning problem in a sensible and logical way where action and state together become a node of MDP [52].

At the start of the learning process, the agent usually has no information about how to act according to the states its receives so it needs to explore which actions result in a maximum reward. One of the important concepts during this exploration process is that the action taken at a given time not only affects the current rewards but the rewards following it which brings us to the concept called *delayed reward*.

The reward returned from the environment is the immediate reward whereas the *return* term is used for cumulative reward till the end of an episode. The aim of the reinforcement learning is to obtain the maximum return, not the reward. Return is defined as,

$$G_t = r_t + r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_T = \sum_{k=0}^{T-t} r_{t+k}, \quad (3.26)$$

where T is the end time step of the episode and t is the current time step. G_t denotes the cumulative reward obtained from the current time step till the end of the episode. Since as we get further away from the current time step, the reward obtained gets more unpredictable, a *discounted* version of the return is defined in which as we get more and more away from the current time, the less influence the reward obtained

affects the return:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^2 r_{t+3} + \dots + \gamma^{T-t} r_T = \sum_{k=0}^{T-t} \gamma^k r_{t+k}, \quad (3.27)$$

where γ is called the discount factor in $[0,1]$ interval and allows a trade-off between the long term and short term rewards.

Another important concept in reinforcement learning is the *value* function, which is the expected value of the discounted return of a state when a policy is followed, can be defined as:

$$V^\pi(s) = \mathbb{E}[G_t | s_t = s, \pi] = \mathbb{E}\left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | s_t = s, \pi\right]. \quad (3.28)$$

Value functions are useful to evaluate a policy since it gives the expected return of a state when the actions are derived from the given policy. So, following this sense, there should be an optimal value function which gives the maximum expected reward under some policy, which is the optimal value function and it is given as:

$$V^*(s) = \max_{\pi \in \Pi} (V^\pi(s)), \quad (3.29)$$

where Π is the set of all possible policies of the system. Just like the value function, there is also one more function that gives the expected return which is called *action-value* function. Though they yield the same result, the difference between them is that the value function, from the start and till the end, follows the policy to choose an action whereas, in action-value function, the first action is not taken from the policy but given as a parameter to the function. After this, action-value function, just like in the value function, follows the policy. The action value function can be defined as:

$$Q^\pi(s, a) = \mathbb{E}[G_t | s_t = s, a_t = a, \pi] = \mathbb{E}\left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi\right], \quad (3.30)$$

and just like the value function, optimal action-value function can be defined as:

$$Q^*(s, a) = \max_{\pi \in \Pi} (Q^\pi(s, a)). \quad (3.31)$$

From Equation 3.29 and Equation 3.31, it is clear that there exist a policy that yields the optimal value function and the action-value function which is called *optimal policy*. The optimal policy is considered as the solution to the reinforcement learning problem since this optimal policy yields the optimal value function and the optimal

action-value function which means that by following this policy, the agent is bound to get the maximum expected reward.

Also, to have a feeling about the difference between following a policy to choose an action or not, the *advantage function* is used. The advantage function is just the difference between action-value function and the value function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \quad (3.32)$$

Last but not least, the accumulation of reward in Equation 3.30 can also be calculated by adding a reward term at each step in a recursive way. This methodology is called the Bellman Equation [53] and can be defined for action value-function as:

$$Q^*(s, a) = \sum_{s' \in S} P(s'|s, a)(R(s, a, s') + \gamma \max_{a' \in A} Q^*(s', a')), \quad (3.33)$$

where P is the state transition defined in Equation 3.19 and R is the reward function that gives the reward signal and depends on current action and state and the resulting state.

There are two main different approaches in reinforcement learning, named tabular methods and function approximation methods [52]. In tabular methods, the values of the states are stored in a tabular way and they are updated by following the formulation of the Bellman Equation. In tabular methods, every state has its equivalent value that is being stored. One example of the tabular method is Q learning [54]. In Q learning, the action value-function is stored in a table that contains a value for every state-action pair. During training, this table is updated for every time step, and it is proved that this method outputs the optimal action-value function eventually [54]. Though tabular methods, in theory, converge to the optimal values, they are more used in low-dimensional problems due to their time and memory constraints. And this brings us to the second method which is called the function approximation method. In the function approximation methods, instead of storing values for every state or state-action pair, a general function like a neural network is used to store the values that approximate the value of a given state or state-action pair. One of the best examples of this method is the Deep reinforcement learning method which we cover in Section 3.4.

3.4 Deep Reinforcement Learning

The deep reinforcement learning method, as the name suggests, is the combination of deep learning and reinforcement learning [55]. Though deep reinforcement learning has some differences with reinforcement learning, the underlying idea still holds, i.e. just like in reinforcement learning, the goal of deep reinforcement learning is to find the optimal value, the action value, or the policy. The biggest difference is that deep neural networks are used in deep reinforcement learning as a function approximator hence the term deep in it. In other words, in deep reinforcement learning, neural networks are used to represent the value function or the policy.

Although there are lots of methods in deep reinforcement learning, in this section the emphasis is going to be on the methods that are used as a basis of the used methods in this thesis. More detailed information is given in Chapter 4 for the methods used in this thesis.

3.4.1 Value Gradient Methods

In value-gradient methods, the deep neural network is used as the function approximator of the value function. So, the neural network takes the states as input and outputs the value of that states. After that, base on the value, a policy is derived. One example of this method is Deep Q Network (DQN) [38] which is covered in this subsection since it can be considered as the basis of some of the algorithms that we use in this thesis.

DQN is an off-policy deep reinforcement learning algorithm that uses a neural network for the action-value function. The resulting policy is derived from the action-value such that the action that makes the action-value function maximum is chosen. Due to this, actions have to be a finite number, i.e., DQN does not work for the continuous action space problems and only works for discrete action space problems. Two main improvements made DQN successful: One of them is the concept of a *target network* which makes the update of the neural network more stable. The idea of the target network is that a second network from the original one is created and this network is used during the training of the original network and at some intervals, it

is synchronized with the original one. The other concept is the addition of *replay buffer*. Due to the structure of the reinforcement learning, there is a correlation between each step which can make the update of the neural network unstable since this does not hold the assumption of i.i.d. (independently and identically distributed). To solve this problem, DQN created the concept of *replay buffer*. During training after every step is taken, four values are stored in a buffer in a form of tuple $\langle s, a, r, s' \rangle$ which are the current state, reward, taken action, and the resulting state. After this, during training or in term of deep learning, during the update of the neural network randomly sampled data from this replay buffer is used which enables to do batch gradient descent for the network update. These two addition is used to increase the stability of the algorithm which is one of the main issues for the deep reinforcement learning [38].

The algorithm of the DQN first initializes the weights without any prior knowledge as θ_0 for the network of action-value function $Q(s, a; \theta_0)$ and create a copy of this network called the target network. After this, it starts the agent to chose an action either from the policy or randomly till sufficient amount of data is acquired in the replay buffer. Later at some defined intervals, the original network is updated using:

$$\theta_{k+1} = \theta_k + \alpha (Y_k^Q - Q(s, a; \theta_k)) \nabla_{\theta_k} Q(s, a; \theta_k), \quad (3.34)$$

where α is the learning rate and Y_k^Q is the target action-values which is defined as:

$$Y_k^Q = r + \gamma \max_{a' \in A} Q(s', a'; \theta_k). \quad (3.35)$$

Equation 3.34 is obtained from the objective loss function stated in DQN [38] as:

$$L_{DQN}(\theta_k) = (Y_k^Q - Q(s, a; \theta_k))^2. \quad (3.36)$$

Finally, after some defined interval the target network is synchronized with the original one. Figure 3.4 shows the high-level block diagram of the DQN approach. Though there are many different variants of the explained algorithm above, since the main idea does not change much, we do not cover them in this thesis.

3.4.2 Policy Gradient Methods

In policy gradient methods, the policy itself directly becomes the neural network that is needed to be optimized. Since, in this thesis, policy gradient methods are used, this

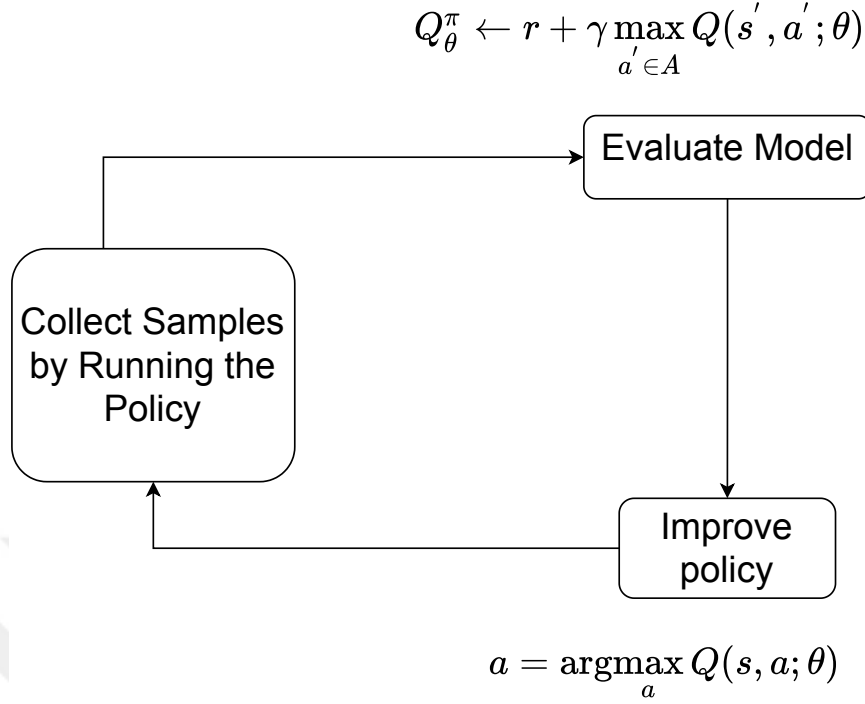


Figure 3.4: DQN block diagram

subsection gives a brief background about the policy gradient-based methods. Further details about the used methods are covered in Chapter 5.

The initial state value can be obtained from Equation 3.28 that is bound to follow a stochastic policy:

$$V^{\pi}(s_0) = \int_S \rho^{\pi}(s) \int_A \pi(s, a) R(s, a) da ds, \quad (3.37)$$

where $\rho^{\pi}(s)$ is the distribution of states with some discount factor when following a policy π :

$$\rho^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t \Pr\{s_t = s | s_0, \pi\}. \quad (3.38)$$

Assume that the neural network base policy has the parameters ϖ . Then, gradient of the policy [39] is :

$$\nabla_{\varpi} V^{\pi_{\varpi}}(s_0) = \int_S \rho^{\pi_{\varpi}}(s) \int_A \nabla_{\varpi} \pi_{\varpi}(s, a) Q^{\pi_{\varpi}}(s, a) da ds, \quad (3.39)$$

and by multiplying and dividing the gradient of policy with π_{ϖ} we obtain:

$$\nabla_{\varpi} \pi_{\varpi}(s, a) = \pi_{\varpi}(s, a) \frac{\nabla_{\varpi} \pi_{\varpi}(s, a)}{\pi_{\varpi}(s, a)} = \pi_{\varpi}(s, a) \nabla_{\varpi} \log(\pi_{\varpi}(s, a)), \quad (3.40)$$

which makes the Equation 3.39 to an expectation as

$$\nabla_{\varpi} \pi_{\varpi}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_{\varpi}}, a \sim \pi_{\varpi}} [\nabla_{\varpi} \log(\pi_{\varpi}(s, a)) Q^{\pi_{\varpi}}(s, a)]. \quad (3.41)$$

Using Equation 3.41, the policy can be updated at each step. In addition to this, an additional network that also tries to estimate the value can be used. This kind of method where both policy and value functions are estimated is called the actor-critic method. The actor is for the policy and is the goal to learn whereas the critic is for the value function and used for criticizing the actor. The overall block diagram for actor-critic architecture can be seen in Figure 3.5. As mention before the further details of each policy gradient algorithms used in this paper are given in Chapter 5.

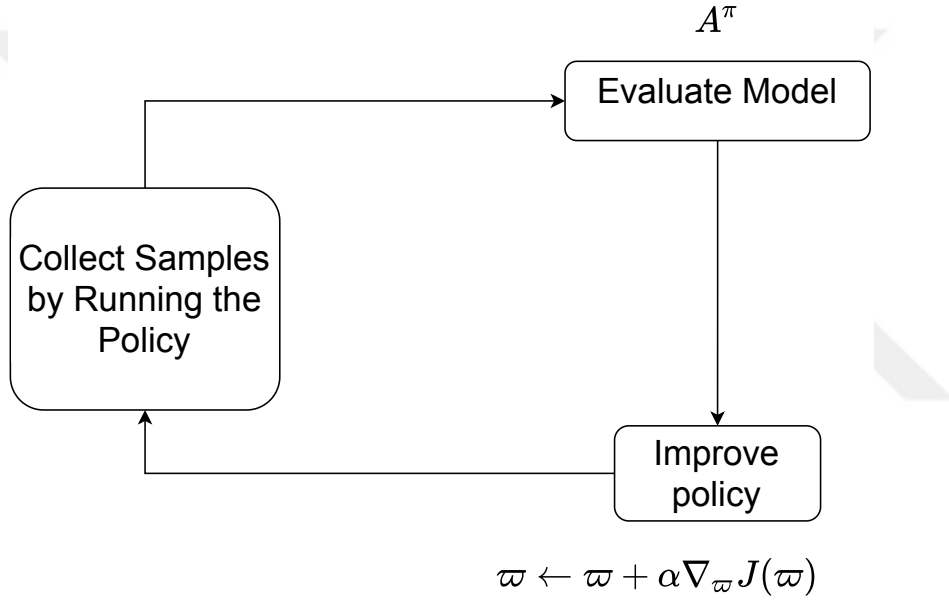


Figure 3.5: Actor-Critic architecture high level block diagram.

CHAPTER 4

METHODS

In this chapter, we give the details of the algorithms implemented in this thesis starting with Section 4.1 where the classical control approach is explained. In Section 4.2 learning-based approaches are given.

4.1 Classical Control Approach

4.1.1 Trajectory Generation

According to Thomas et al. [2], a system can be considered flat if a coordinate change exists such that the states and the control input of the system can be written in terms of the flat outputs and their derivatives. Moreover, if this change of coordinates is a *diffeomorphism*, the flat outputs $\mathbf{y}$ and their flat space derivatives can be used for trajectory planning due to the fact that there is a unique mapping from the full state space of the system and the flat outputs of the aerial manipulator which is differentially flat with the given flat output set [2] (for details, see [18]).

$$\mathbf{y} = [x, y, \beta]^T. \quad (4.1)$$

The trajectories are divided to piecewise polynomials as n by m order where n is the number flat outputs variables and m is the number of polynomials. The cost function created from the snap of the trajectory is defined as:

$$J_i = \int_{t_0}^T \left\| y_i^{(4)}(t) \right\|^2 dt, \quad (4.2)$$

where $y_i(t)$ denotes the i^{th} flat output at $m = t$. This trajectory can be redefined in the form of Quadratic Program (QP) [18] as:

$$\min c^T H c + f^T c \quad (4.3)$$

$$\text{s.t } A c \leq b$$

where c is the 3nm by 1 vector of the flat outputs $y_i(t)$. QP programming can be used to find the trajectories with Equation 4.3.

4.1.2 Conventional Controller Design

The controllers for the aerial vehicle and the manipulator are considered separately. The controller for the aerial vehicle design is based on Ugurlu et al. [17]. The controller named "conventional controller" consists of angular and positional control of the aerial vehicle. The bodies of the aerial vehicle and the manipulator are handled separately. For the aerial vehicle case the control inputs are left and right motor thrust. These thrust forces f_l and f_r can be seen in Equation 3.5 affects the translational motion of the vehicle. The torque is also generated from the difference of these two forces, as shown in Equation 3.15. This generated torque affects the rotational motion of the aerial vehicle, as seen in Equation 3.11.

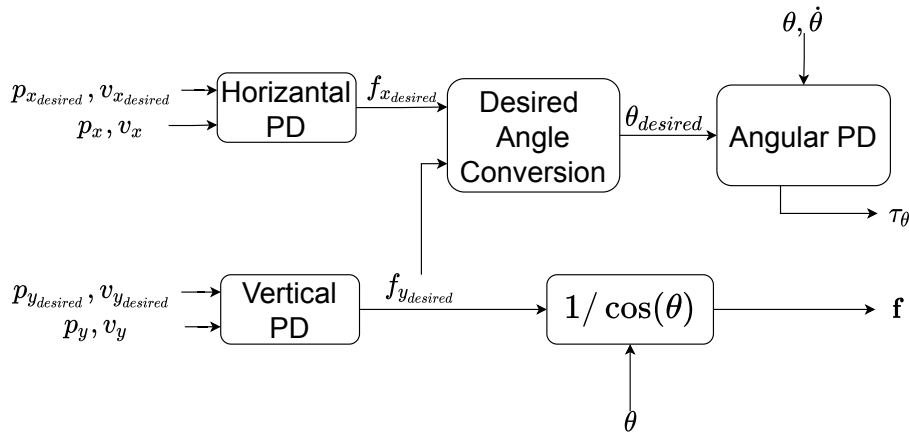


Figure 4.1: Conventional controller block diagram.

Figure 4.1 shows the block diagram for the aerial vehicle where $p_{x_desired}, v_{x_desired}$ are the desired position and velocity that are generated from the trajectory generation al-

gorithm in x -direction of the inertial frame and $p_{y_{desired}}, v_{y_{desired}}$ are in the y -direction of the inertial frame. Controller takes the position and velocity input generated from the trajectory generation algorithm defined in Section 4.1.1 and the observed position and velocities to generate the required torque and forces for the aerial vehicle to follow the trajectory. In Figure 4.1, f is the total force, such that:

$$f = f_l + f_r, \quad (4.4)$$

and the term τ is given in Equation 3.15. From Equation 3.15 and Equation 4.4, f_l and f_r can be taken as:

$$f_l = \frac{f}{2} + \frac{\tau}{2d}, \quad (4.5)$$

$$f_r = \frac{f}{2} - \frac{\tau}{2d}. \quad (4.6)$$

Moreover, Horizontal PD and Vertical PD blocks are Proportional-Derivative (PD) controllers that in horizontal and vertical direction respectively. They generate the net force in their respective directions in inertial frame that can be seen in Figure 3.1. Therefore, the desired force in x -direction of the inertial frame $f_{desired_x}$ is defined as:

$$f_{desired_x} = -(K_{p_x}(p_{x_{desired}} - p_x) + K_{d_x}(v_{x_{desired}} - v_x)), \quad (4.7)$$

and the desired force in y -direction of the inertial frame $f_{desired_y}$ is defined as,

$$f_{desired_y} = -(K_{p_y}(p_{y_{desired}} - p_y) + K_{d_y}(v_{y_{desired}} - v_y)), \quad (4.8)$$

where all K_p and K_d are controller parameters.

In addition to these blocks, the Desired Angle Conversion block is used to find the desired angle to create these forces since the forces f_l and f_r are in body frame of the aerial vehicle, angle θ is needed to convert them to the inertial frame which is given in Equation 3.10. Therefore, mathematically, the job of this block is defined as:

$$\theta_{desired} = \text{clip}(\text{atan2}(f_{desired_x}, f_{desired_y}), \{\theta_{max}, \theta_{min}\}), \quad (4.9)$$

where $\text{clip}(x, range)$ function simply clips the desired θ angle according to the given limits θ_{max} and θ_{min} .

The fourth block, Angular PD, is the proportional-Derivative controller that is used for controlling the attitude of the aerial vehicle with the torque command τ_θ is defined

as:

$$\tau_\theta = K_{p_\theta}(\theta_{desired} - \theta) + K_{d_\theta}\dot{\theta}, \quad (4.10)$$

The manipulator control on the other hand is a simple Proportional controller that

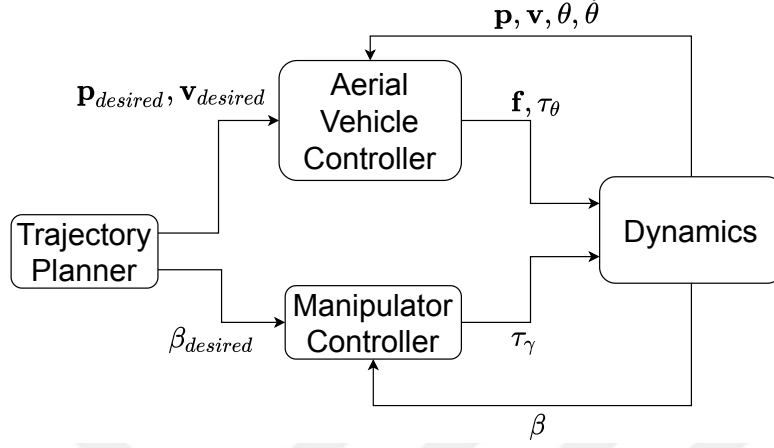


Figure 4.2: Aerial Manipulator control diagram.

takes the desired β angle from the trajectory generation algorithms and outputs the actuator command which is speed which can be seen in Figure 4.2 that shows the overall controller diagram of the aerial manipulator.

4.2 Learning-based Approaches

Actor-critic algorithms are considered one of the most used state-of-the-art deep reinforcement learning algorithms. They use both the value function and the policy to benefit from both. The actor part is composed of the policy that is required to take the action and the critic part, which can either be a value or action-value function, is used to evaluate the quality of the actor.

All of the proposed approaches in this thesis follow the actor-critic architecture and they can be grouped together under two main groups: on-policy algorithms, and off-policy algorithms. In on-policy algorithms, the policy used in training and the evaluation is the same. This usually means that, at the start of the training, the policy is usually ‘soft’, which means every action has a probability to be chosen by the policy. However, during training, the policy gradually gets closer and closer to be a deter-

ministic optimal policy. The policy needs to be a soft one at the start of the training due to a well-known phenomenon in reinforcement learning: the trade-off between exploration and exploitation. The agents need to explore different actions to take better actions but as it gets closer to the optimal policy, it needs to exploit what it already knows. Therefore, on-policy algorithms try to solve this problem by creating a soft policy at the start and gradually move towards a deterministic one so, on-policy algorithms at the end, do not learn an optimal policy but a policy very close to it. Off-policy algorithms, however, choose another approach. Off-policy algorithms use two different policies in the algorithm, named target policy and a behavior policy. Behavior policy is used during training to explore and is used as the policy to interact with the environment. Target policy on the other hand is the policy that the agent tries to learn. So, in off-policy, older policy-generated data can be used and said to explore better than the on-policy ones due to their nature. In this study, we make a comparison between on-policy and off-policy algorithms for the aerial manipulator control. We use *stable-baselines* [56] package for the implementation of the learning based algorithms.

4.2.1 On-policy Algorithms

The used on-policy algorithms in this thesis are Trust Region Policy Optimization (TRPO) [14], a deterministic synchronous version of Asynchronous Advantage Actor Critic (A3C) method [15] and Proximal Policy Optimization (PPO) [16].

4.2.1.1 Trust Region Policy Optimization

In Trust Region Policy Optimization (TRPO) method, a modified version of the natural policy gradient is used such that the update in policy is restricted with Kullback Leibler (KL) divergence. More specifically:

$$\max_{\varpi} \mathbb{E}_{s \sim \rho^{\pi_{\varpi}}, a \sim \pi_{\varpi}} \left[\frac{\pi_{\varpi}(s, a)}{\pi_{\varpi_{old}}(s, a)} A^{\pi_{\varpi}}(s, a) \right], \quad (4.11)$$

which is subject to:

$$\mathbb{E} [D_{KL}(\pi_{\varpi_{old}}(s, \cdot) || \pi_{\varpi}(s, \cdot))] \leq \delta, \quad (4.12)$$

where δ is a hyper-parameter and ϖ_{old} is the parameter just before the update. Instead of the restriction, the theory of TRPO proposes a penalty, with which the optimization of solving the unconstrained problem becomes:

$$\max_{\varpi} \mathbb{E}_{s \sim \rho^{\pi_{\varpi}}, a \sim \pi_{\varpi}} \left[\frac{\pi_{\varpi}(s, a)}{\pi_{\varpi_{old}}(s, a)} A^{\pi_{\varpi}}(s, a) - \beta D_{KL}(\pi_{\varpi_{old}}(s, \cdot) || \pi_{\varpi}(s, \cdot)) \right], \quad (4.13)$$

which is the optimization problem solved by TRPO. The term β is also a hyper-parameter.

4.2.1.2 Asynchronous Advantage Actor Critic

To solve the instability of the gradient-based approach in deep reinforcement learning, the Asynchronous Advantage Actor-Critic algorithm uses parallel learners instead of using a replay buffer [15]. The execution of multiple learning agents on multiple instances of the environment in a parallel way not only makes the data closer to i.i.d. since parallel agents most likely will experience different states, but also enables the usage of CPUs (Central Processing Units) for learning instead of GPUs (Graphics Processing Units) which provides practical benefits.

Dhariwal et al. [57] found that instead of an asynchronous one, a synchronous one where each agent waits for the agent to move around the required amount of states before the update reduces the noise provided by the asynchrony of the algorithm. This method is called simply Advantage Actor-Critic (A2C) [58] which is the version of the A3C used in this paper.

The A3C method uses neural network for both the policy function $\pi_{\varpi}(s, a)$ and the value function $V_{\varpi_v}(s)$. Moreover, the actor and the critic are updated after enough steps have passed. The update function for the A3C algorithm is defined as:

$$\nabla_{\varpi} \log \pi_{\varpi}(s, a) (A_{\varpi, \varpi_v}(s, a)) + \beta \nabla_{\varpi} H(\pi_{\varpi}(s, a), \quad (4.14)$$

where β is a hyper-parameter and $A_{\varpi, \varpi_v}(s, a)$ is the estimation of the advantage function given as:

$$\sum_{k=0}^{i-1} \gamma^k r_{t+k} + \gamma^i V_{\varpi_v}(s_{t+i}) - V_{\varpi_v}(s_t), \quad (4.15)$$

where i can change for states but it is upper-bound by the required steps for the update. The term $H(\pi_\theta(s, a))$ is simply entropy that is used for increasing the exploration by making the policy less deterministic.

4.2.1.3 Proximal Policy Optimization

Proximal Policy Optimization [16] is a method based on TRPO with some modifications done on the objective function with the addition of the parallel agent usage [15] which results in an easier to implement method that also yields better results for a number of problems. The idea in the PPO is similar to that in TRPO such that the output of the update of the policy should not be too far away from the policy before the update, to increase the stability of the algorithm. To achieve this, a clipped surrogate objective function is defined in the PPO algorithm that is used for updating the agents' parameters. If probability ratio $r_t(\varpi)$ is defined as:

$$r_t(\varpi) = \frac{\pi_\varpi(s, a)}{\pi_{\varpi_{old}}(s, a)}, \quad (4.16)$$

where $r_t(\varpi_{old}) = 1$, the clipped objective function that is different from the KL-divergence penalty can be defined as:

$$\mathbb{E}_{s \sim \rho^{\pi_\varpi}, a \sim \pi_\varpi} [\min(r_t(\varpi)A^{\pi_\varpi}(s, a), \text{clip}(r_t(\varpi), 1 - \epsilon, 1 + \epsilon)A^{\pi_\varpi}(s, a))], \quad (4.17)$$

where the second term $\text{clip}(r_t(\varpi), 1 - \epsilon, 1 + \epsilon)$ is used for clipping the probability ratio so that the result of the update does not move away too much from the value before the update. Note that the term ϵ is another hyper-parameter.

In addition to these, just like in the A3C algorithm, the PPO algorithm also uses an entropy term in the update function to further reduce the big jumps and to increase the exploration.

4.2.2 Off Policy Algorithms

The used off-policy algorithms in this thesis are Deep Deterministic Policy Gradient (DDPG) [11], Twin Delayed Deep Deterministic Policy Gradient (TD3) [12], Soft Actor-Critic (SAC) [13].

4.2.2.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [11] is an actor-critic, model-free algorithm that is used for continuous space problems and is based upon the deterministic policy gradient theorem [40] and DQN [38]. DDPG uses an experience replay buffer and a target network, which are based on DQN, in the deterministic policy gradient actor-critic architecture to increase the stability of the algorithm.

The value function update objective is defined as:

$$L_{\varpi_v} = \mathbb{E}[L_{DQN}(\varpi)], \quad (4.18)$$

where L_{DQN} is defined in Equation 3.35. And the update function of the actor is defined as,

$$\mathbb{E}_{s \sim \rho^{\pi_{\varpi}}, a \sim \pi_{\varpi}} [\nabla_{\varpi} \pi_{\varpi}(s, a) \nabla_a Q_{\varpi_v}(s, a)]. \quad (4.19)$$

Since DDPG is an off-policy method, a stochastic policy is used in order to get experience during training which is different from the target policy. The exploration policy is usually achieved by directly adding noises like Gaussian or Ornstein–Uhlenbeck noise [59] to the output of the policy.

4.2.2.2 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed Deep Deterministic Policy Gradient (TD3) [12] is built upon Double Q-learning, a variant of Q-learning, and DDPG [11] to solve the problem of overestimation of action-value function that results in sub-optimal policy in the actor-critic architecture. In the TD3 algorithm, a pair of independent critics are trained. This approach is similar to Double Q learning but a clipped version of it. In TD3, the less-biased critic estimate is upper-bounded by the other estimate that results in choosing the minimum between the critics as the target update given as:

$$y_1 = r + \gamma \min_{i=1,2} Q_{\varpi_{v_i}}(s, \pi_{\varpi}(s, a)). \quad (4.20)$$

TD3 also relies on delayed target network update and addition of noise to the target action when computing the value of Equation 4.20. The objective function of TD3 is the same as in the DDPG algorithm which is given in Equation 4.19 and Equation 4.18 with the difference explained in Equation 4.20 and addition of noise to the action.

4.2.2.3 Soft Actor Critic

Just like in the TD3 algorithm, the Soft Actor-Critic [13] algorithm also relies on the clipped version of double Q-learning but differs from TD3 in terms of policy in that the SAC algorithm tries to optimize a stochastic policy instead of a deterministic one. The soft actor-critic method can be considered as a combination of actor-critic methods and maximum entropy reinforcement learning. In max entropy reinforcement learning, entropy is used as a term to increase exploration and in SAC this term is added in objects such as:

$$\mathbb{E}_{s \sim \rho^{\pi_{\varpi}}, a \sim \pi_{\varpi}}[G_t + \alpha H(\pi_{\varpi}(s, a))], \quad (4.21)$$

where α is used for controlling the stochasticity of the optimal policy and entropy H is defined as

$$H(P) = \mathbb{E}_{x \sim P}[-\log P(x)]. \quad (4.22)$$



CHAPTER 5

SIMULATION ENVIRONMENT AND SIMULATION DETAILS

In this chapter, details about the simulation environment, which is the environment used for the training of the learning-based algorithms and for the optimization of the conventional controller, are given. In Section 5.1 details about how simulation environment is designed given an in Section 5.2, information about the environment variables are given.

5.1 Design of the Simulation Environment

We have designed a simulation environment to simulate the aerial manipulator dynamics for a pick and place task. The deep learning algorithms interact with the designed environment as seen in Figure 3.3. The simulation environment, in its most basic explanation, at a time step t gets the action signal a_t as an input, simulates the dynamics of the aerial manipulator for a small duration Δt and outputs the next state s_{t+1} with the addition of the reward signal r_{t+1} according to the dynamics of the system. The dynamics that enables the passage from s_t to s_{t+1} under some given action signal is explained in Chapter 3. The resulting reward signal value calculation is explained in detail in Section 5.5.

For both the implementation of the simulation environment and the methods explained in Chapter 4, Python programming language is used. For the simulation environment, Box2D physics engine [60] with the addition of OpenAI's gym [61] toolkit that enables the environment to interact with the algorithms, are used in this thesis. Box2D physics engine is an open-source physics engine that is used for solving the differential equations of 2D rigid bodies. On the other hand, the gym toolkit

is a very well-known deep reinforcement learning interface that is used in this field for comparison of algorithms.

The simulation environment of this thesis is based on the environment created by Ugurlu et al. [17] in which a point mass representation of the aerial vehicle was created in Box2D. By using the model created by Ugurlu et al. [17], we have created a shaped body for the aerial vehicle instead of a point mass representation of it, added joints between the manipulator and aerial vehicle as well between the gripper and the manipulator. In addition to these, we have added contact detection functions according to the Box2D documentation to recognize the contact between the bodies.

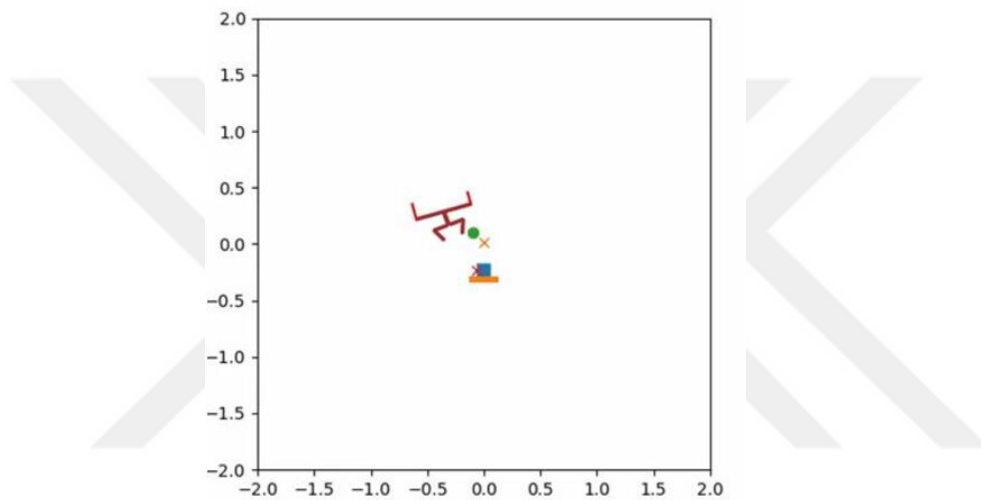


Figure 5.1: An instance of Aerial manipulation for pick and place operation. The green dot is the goal position of the box which is the blue squared shape object. The dark red line is the Aerial manipulator and the light red lines indicate rotor thrusts. The red cross is the aim of the left gripper end-point location and the light yellow cross is the goal position of the aerial manipulator. The yellow line under the blue box can be considered as ground.

The gym interface consist of four methods named *init*, *step*, *reset* and *render*. Since the deep reinforcement learning algorithms look into the two variables in order to get to know the state space and action space, in *init* method in two variables that are named *action_space* and *observation_space* with their types given, must be created. In our case, both of these variables had Box type which is the type for continuous space that is explained in the documentation of the gym toolkit. Moreover, the limits

of each space variable are also must be written in this function. The *reset* method is used for initiating the environment as the name suggests. So, the initial state of the environment is decided in this method according to the initial state distribution, which is given in Section 5.3 for our case. The *step* method on the other hand can be considered as the backbone of the interface. This method is called to take a step on the environment based on the action it takes and the current state and produces the next state and the reward signal with it. In addition to these, the termination status and additional information are also provided for the algorithm too. The reward signal function is described in detail in Section 5.5. The last method of the interface is *render* which is used for rendering the environment. Though this method is not a must, seeing the process of the environment can be help full.

The first goal of the aerial manipulation task is the stabilization of the aerial manipulator, then the aerial manipulator has to reach the box object and grab it, after that the aerial manipulator has to move the object to the given location. The task finishes when the box reaches the goal position otherwise task will continue till the amount of time that given for this task passes. An instance of this task can be seen in Figure 5.1.

5.2 States and Actions of the Simulation

The state space and the action space of the environment are continuous. The states are considered to be fully observable and consist of:

$$s_t = [p_{b_x}, p_{b_y}, p_{p_x}, p_{p_y}, p_{g_x}, p_{g_y}, \psi, \theta, \gamma, \beta, v_{p_x}, v_{p_y}, \dot{\theta}, G], \quad (5.1)$$

where G is a special state that represents whether the gripper gripped the box or not. The terms p_{b_x} and p_{b_y} are used as the positions of the box x-direction and y-direction in inertial frame respectively. Likewise, p_{p_x} and p_{p_y} terms define the position of the platform in inertial frame and p_{g_x} and p_{g_y} are used as the position of the left gripper end position in inertial frame. The ψ is the orientation of the box object, θ is the orientation of the aerial vehicle and γ is the angle between the left gripper and the manipulator whereas β is the angle between the manipulator and aerial vehicle.

The actions, on the other hand, consist of,

$$a_t = [f_l, f_r, v_{j_{arm}}, v_{j_{gripper}}], \quad (5.2)$$

where f_l, f_r correspond to left and right thrusts in terms of force, and $v_{j_{arm}}, v_{j_{gripper}}$ correspond to manipulator main joint and gripper joint commands in terms of speed.

5.3 Initial State Constraints

The initial position of the aerial manipulator is a uniform distribution over a state space within the scope of the problem. Mathematically ,

$$p_{p_x} \sim U(-1, 1), \quad (5.3)$$

$$p_{p_y} \sim U(0.2, 1). \quad (5.4)$$

The starting point of the box is fixed and defined as:

$$p_{b_x} = 0, \quad (5.5)$$

$$p_{b_y} = -0.25. \quad (5.6)$$

5.4 Termination Criteria

There are four termination cases during the training of the learning approaches:

- First criteria: If the box reaches its destination, the episode finishes. Box reaches the goal when it gets in a circular region of the goal position that has a radius of 0.1.
- The second criteria is the time limit of the episodes. The given time step for each episode is 400 which is equal to 4 seconds since the simulation runs at 100Hz. If this amount is reached, the episode is terminated.
- The third termination criteria is when the aerial manipulator moves out of the determined task space, which 4 by 4 square space.
- The last termination condition is very similar to the third one but instead of the aerial manipulator, this criteria looks for whether the box is out of the determined task space or not.

5.5 Reward Shaping

In reinforcement learning, the reward is usually obtained at the end of the process, which is called sparse reward. Reward shaping, on the other hand, is used for dividing this sparse reward to each state to obtain more info in order to reduce the exploration duration of the problem. In this thesis, the reward obtained at a time step, r_t , is a combination of the approaches by Ugurlu et al. [17], Li et al. [48] and Popov et al. [4]:

$$r_t = \begin{cases} (T-t)5, & \text{if goal reached} \\ (T-t)P_c, & \text{if crash occurs} \\ (T-t)P_o, & \text{if out of border} \\ k_{p_g} \|e_{gripper}\| + k_{p_b} \|e_{box}\| + k_v \|v_t\| + k_\theta \|\theta\| + k_a \|a_t\| + k_g(G) & \text{otherwise.} \end{cases} \quad (5.7)$$

Table 5.1: Reward function parameters

Parameters	Value
P_c	0.75
P_o	2.5
k_{p_g}	1.5
k_{p_b}	2
k_v	0.2
k_θ	0.5
k_a	0.05
k_g	2.5

where T represents the maximum time step given for one episode. The term $e_{gripper} = [e_{x_{gripper}}, e_{y_{gripper}}]^T$ is defined as the distance between the gripper goal position which can be seen in Figure 5.1 and the gripper left side position. Only one of the grippers is considered because the action command for both of them is the same; that is, both grippers move at the same command but in opposite directions. Box distance error, $e_{box} = [e_{x_{box}}, e_{y_{box}}]^T$, represents the distance between the goal of the box and the current box position. v_t denotes velocity of the aerial manipulator and is a vector

defined as $[v_x, v_y]^T$ and a_t is the action vector defined in Equation 5.2. The G term comes from state space in Equation 5.1 that states if the box is gripped or not. Finally, the P terms and k terms are the weight of these errors defined in Table 5.1. Note that the main issue of the task at hand is moving the box to the goal position. Therefore, positions have more importance than the other errors.



CHAPTER 6

EXPERIMENTS AND RESULTS

In this chapter, we give details about the experiments we conducted and the result of these experiments. In Section 6.1, implementation and optimization details of the conventional controller and deep reinforcement learning are given. In Section 6.2 a comparison between deep reinforcement learning algorithms is conducted. Furthermore, in Section 6.3 comparison results of the deep reinforcement learning and the conventional controller are given. Finally, in Section 6.4 experiment results of effects of different factors on the deep reinforcements algorithms are given.

6.1 Implementation and Optimization Details

This section gives details about the implementation of deep reinforcement learning algorithms and the optimization of the controller approach.

6.1.1 Optimization of Classical Controller Approach

The parameters of the conventional controller that is described in Section 4.1 are optimized with two optimization techniques described by Ugurlu [17]. the first optimizer, Particle Swarm Optimization (PSO) [62] is used for the global optimization of the parameters of the controller, and for fine-tuning Nelder-Mead [63] optimization algorithm is used. To have a fair comparison between the conventional controller and the learning approaches, the reward function described in Equation 5.7 is used. We have conducted the optimization process using the ten random starting positions shown in Figure 6.1.

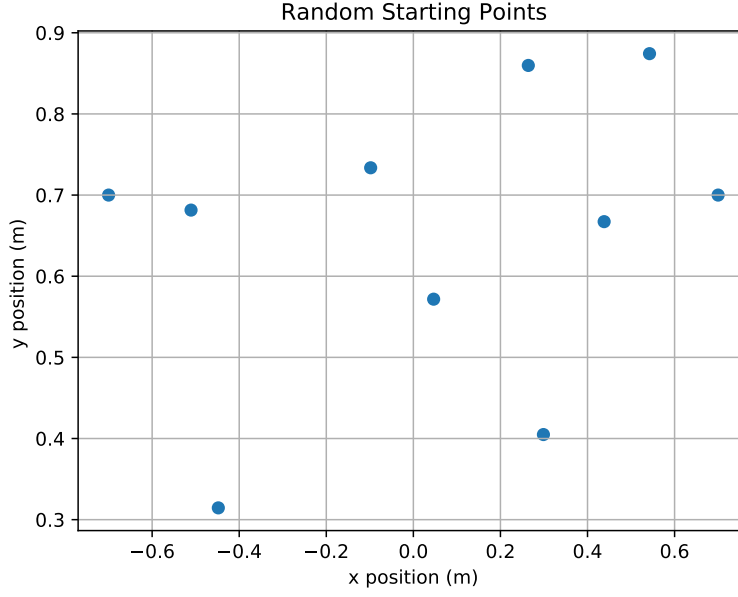


Figure 6.1: Random Starting points for Controller Optimization

We have created a trajectory for each point using similar boundary conditions to reach the destination in a smooth and fast way. In all trajectories other than position and velocity constraints rest of the states do not have constraints and are continuous. Though the QP generates the trajectory fast enough to be able to implement in real-time, in our experiments, we have created the trajectories off-line using Matlab [64] QP function to be able to re-use the same trajectory multiple times for optimization. The resulting parameters obtained after optimization can be seen in Table 6.1.

We present one sample run of the conventional controller with trajectory generation which is initiated from point $p_x, p_y = (0.7, 0.7)$ and observed for 400 timesteps.

In Figure 6.2, the generated trajectory and the observed trajectory can be seen in terms of position. The effects of ignoring the disturbances of the manipulator can be seen when the position gets close to the box as the aerial manipulator drifts a little bit from the generated trajectory since in this region gripping process is occurring. But even if the disturbances are ignored, the observed positions are still close to the generated trajectory and finish the task.

Figure 6.2 also shows the manipulator desired and observed angle. Just like in the

Table 6.1: Optimized controller parameters

Parameter	Values
K_{p_x}	74.352
K_{d_x}	19.014
K_{p_y}	29.425
K_{d_y}	33.813
K_{p_θ}	34.386
K_{d_θ}	4.149
K_{p_γ}	30.301

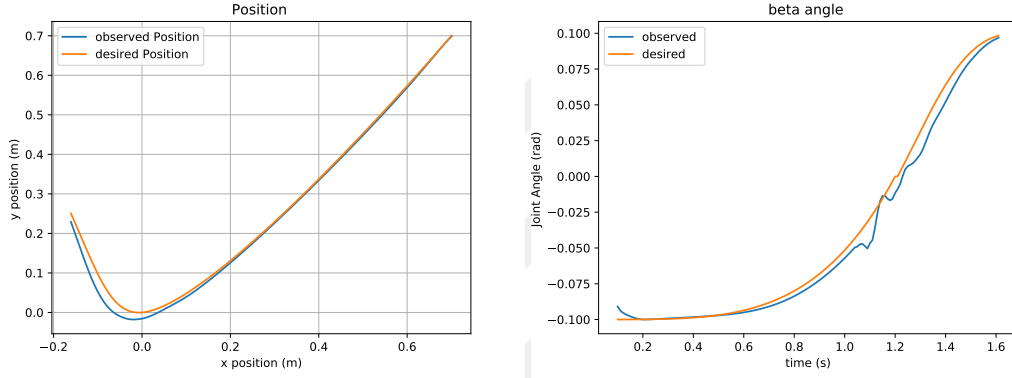


Figure 6.2: Desired and observed trajectory of positions (left) and manipulator arm angle (right). Blue line represents the observed position of the aerial vehicle and the brown line the trajectory generated by using the minimum snap trajectory algorithm.

position case the observed trajectory differs during the gripping process but still, they are close enough to carry out the task at hand.

Velocities of the generated trajectory and the observed ones for x-direction and y-direction are plotted in Figure 6.3. As the plots show, the pattern with the position and the β angle also continues here but the control of velocity is a bit worse than the position which makes sense since in the reward function cost of position error is much bigger than the velocity.

So overall, though the disturbances are not considered during the design of the controller, optimization of the parameters with the current design makes the system robust enough that the system is able to perform the pick and place operation much

faster than the required time given for the task.

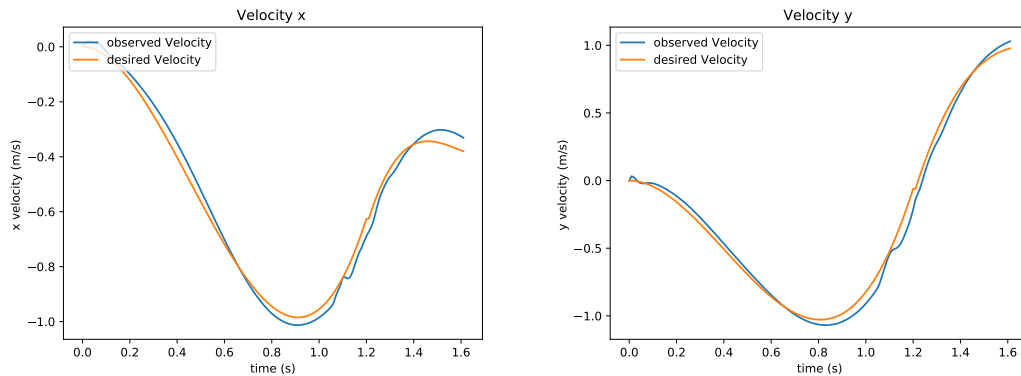


Figure 6.3: Desired and observed trajectory of velocity in the x-direction (left) and y-direction(right). The blue line represents the observed velocity of the aerial vehicle and the brown line represents the trajectory generated by using the minimum snap trajectory algorithm.

6.1.2 Deep Reinforcement Learning Approaches Implementations

For the implementation of the deep reinforcement learning algorithms, stable-baseline [56] is used. Stable-baseline is an open-source package where most of the DRL algorithms are implemented and integrated with the gym environment.

All of the algorithms we used in this thesis have the same 128 by 128 network architecture and the hyper-parameters are kept as the default values given by the stable-baseline. For the algorithms PPO and A2C where the use of multi-environment is encouraged, 4 multi-environment are used.

6.2 Analyzing Learning-based Controllers

In this part of the experiments, the learning-based algorithms are analyzed for solving the pick and place operation for the aerial manipulator and they are compared with each other.

6.2.1 Analyzing On-policy Algorithms

The on-policy deep reinforcement learning algorithms TRPO, PPO, and A2C algorithms are trained for nine million steps and their mean episode reward results of the last 1000 episodes are computed and plotted in Figure 6.4. Other than the A2C, PPO and TPPO learned very valuable policies to control the aerial manipulator for the pick and place operation. PPO learned faster than the TRPO algorithm but no matter how small their final mean reward difference, TRPO obtained a better overall result than PPO. A2C, on the other hand, is worse than both in terms of both speed and convergence.

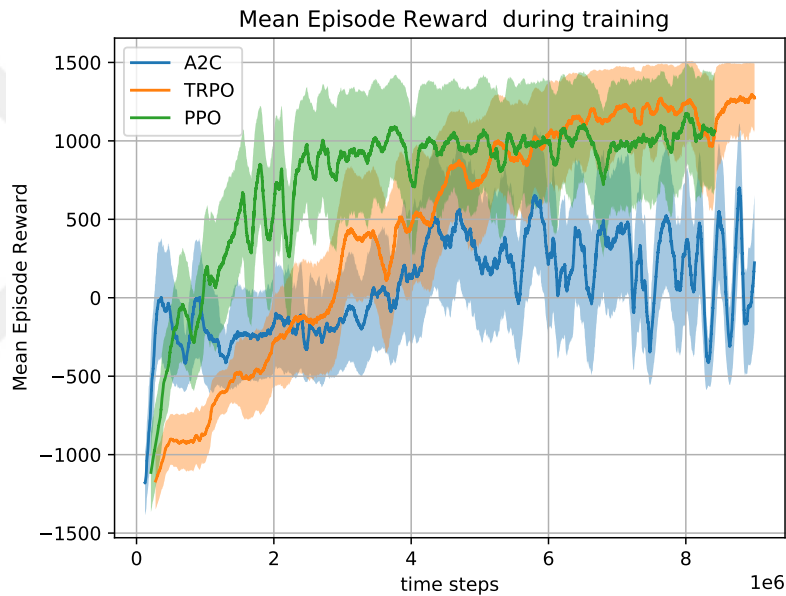


Figure 6.4: Mean episode reward of on-policy algorithms during training. Dark lines represent the mean episode reward and the light regions shows the standard deviation.

The left figure of the Figure 6.5 shows the mean episode length of the three algorithms during training for the last 500 episodes. The figure suggests that all methods achieved a similar mean episode length at the end of the training, with TRPO being the most stable. This means that the speed of all algorithms to solve the pick and place operation is very similar and does not change that much if the initial starting position changes.

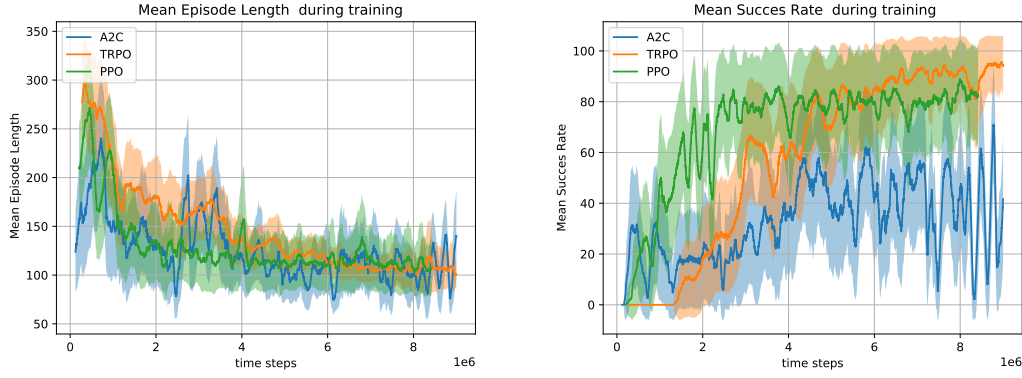


Figure 6.5: Mean episode length (left) and mean episode success rate (right) of on-policy algorithms during training. Dark lines represent the mean episode reward and the light regions show the standard deviation.

In addition to these, the right figure of Figure 6.5 shows the mean success rates of the on-policy methods during the training of the last 500 episodes. Just like in the mean reward case, PPO achieves high success rates faster than any other algorithm, which agrees with the mean reward plot in Figure 6.4. Though PPO achieves a higher success rate faster, TRPO converges to best success rates at the end of the training. From this, we can say that again, different starting points do not affect the success rate much. A2C on the other hand fails to converge to an acceptable success rate.

6.2.2 Analyzing Off-policy Algorithms

For off-policy algorithms DDPG, TD3, SAC, we have conducted training of 3 million time steps and plotted the graphs for the last 1000 episode to see the results better since all of the algorithms gave a similar result as it can be seen in Figure 6.6 that shows the mean episode reward. In this figure, it can be observed that all of the algorithms reach a valuable reward nearly at the same time and the SAC and TD3 algorithm converges to a very similar reward at the end. DDPG converges to a little bit worse mean reward compared to the others.

The left figure of Figure 6.7, on the other hand, shows the mean episode length of the off-policy algorithms. Just like in the mean episode reward case, they all reach a similar mean episode length during training with the SAC algorithm being the best

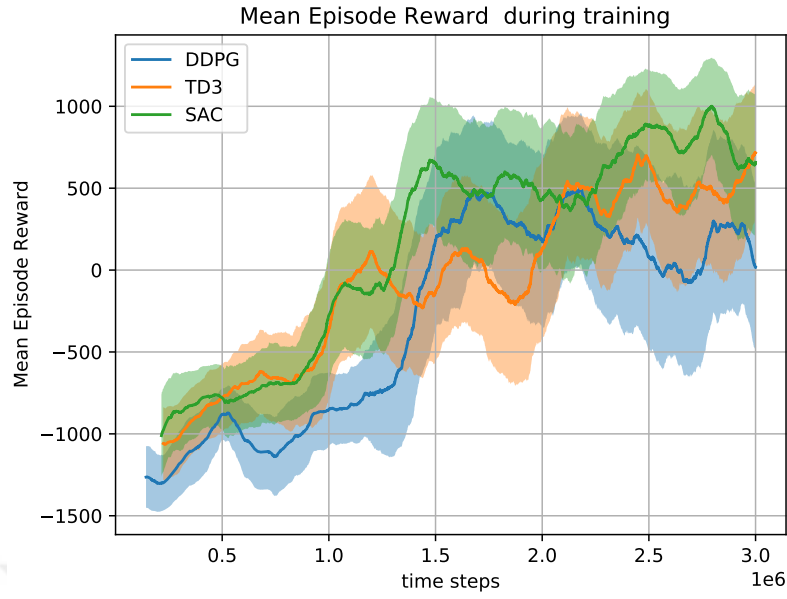


Figure 6.6: Mean episode reward of off-policy algorithms during training. Dark lines represent the mean episode reward and the light regions shows the standard deviation.

and the DDPG algorithm being the worst, which is expected since SAC and TD3 build upon DDPG and try to increase its convergence and stability [12, 13].

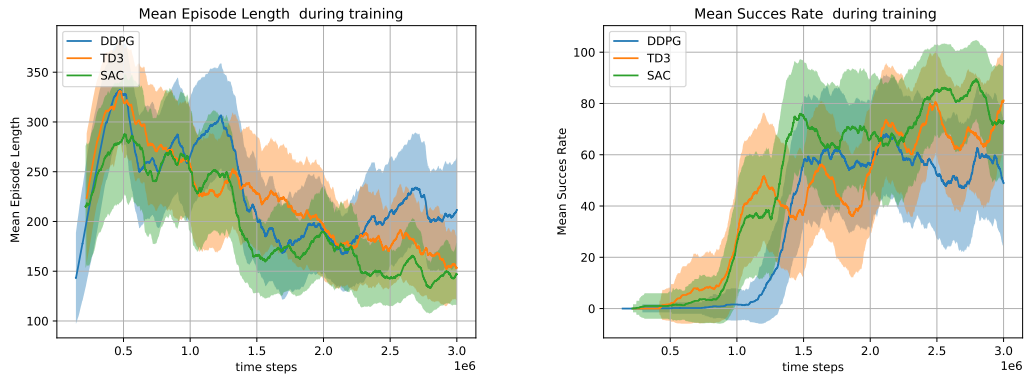


Figure 6.7: Mean episode length (left) and mean episode success rate (right) of off-policy algorithms during training. Dark lines represent the mean episode reward and the light regions shows the standard deviation.

Moreover, the right figure of the Figure 6.7 shows the mean success rates of the three off-policy algorithms during training and again results are the same that SAC and

TD3 achieve very similar success rates and DDPG achieves the worst and all of them achieves a reasonable success rate nearly at the same time but with DDPG being a little bit slower.

6.2.3 On-policy vs. Off-policy Algorithms

In this part of the experiments, we compare the performances of the best off-policy and on-policy models. Plots in the Figure 6.8 show that the converges of TRPO is faster than SAC and the converged value of the best on-policy method, which is TRPO, is better than the SAC algorithm. This might be the case because of not fine-tuning the SAC algorithm's every hyperparameter to the task at hand since the difference between them is not that large.

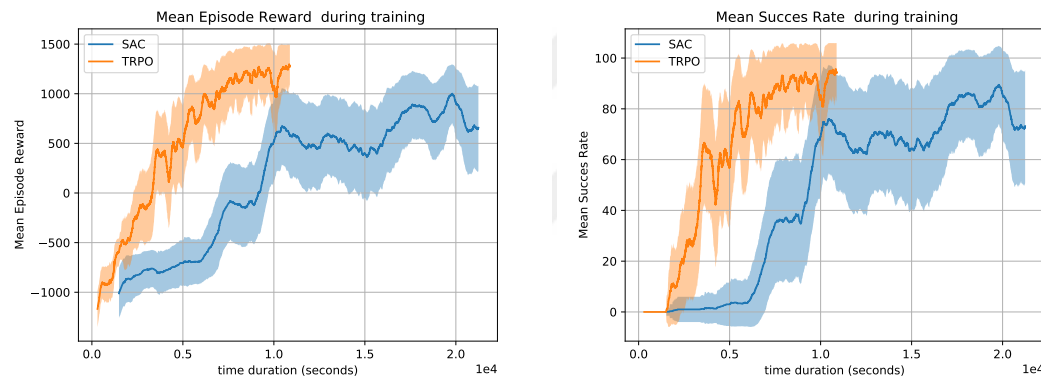


Figure 6.8: Off-policy (SAC) and on-policy (TRPO) comparison graph that shows the Mean episode reward (left) and mean success rates (right) . Dark lines represent mean whereas the light regions represents the standard deviation.

Although, in terms of time duration TRPO is better, this not the case when the graphs are presented in terms of the number of time steps which can be seen in Figure 6.9. SAC algorithm converges to the fairly good reward with much fewer time steps compared to TRPO such that if the same amount of time steps is used for training of the TRPO algorithm, it would have not converged to a reasonable amount of mean reward at all.

Moreover Table 6.2 shows how much time and sample is needed for each algorithm for training. We can see that off-policy algorithms are much more sample efficient



Figure 6.9: Off-policy (SAC) and on-policy (TRPO) mean reward comparison in terms of time steps.

6.3 Learning-based vs. Conventional Control Comparison

Comparison between the best models of all of the on-policy and off-policy algorithms and the conventional controller is tabulated in Table 6.3 in terms of success rates, standard deviation, episode length, and one step execution speed. Note that all of the experiments are conducted on the same computer with a CPU for a fair comparison.

For the learning-based algorithms, 10 trials with each trial containing 500 random points are conducted to obtain the success rates, standard deviation, average episode length, and one-step execution time average. Figure 6.11 shows one of the trials for the on-policies and Figure 6.12 shows the off-policy results for again one of the trials. For the conventional controller, only 10 points are used to obtain the results since we know that from every point in a given space, it is possible to create a trajectory that will perform the task at hand.

As expected, the conventional controller is able to perform the task from every point, that is we can create a trajectory from every starting point and follow it with an optimized controller but it is slower compared to the learning algorithms. Deep rein-

Table 6.2: Training computation results for on-policy and off-policy algorithms

Algorithms	Type	Total Sample (million)	Total Time (seconds)
TRPO	on-policy	9.0	10879.74
A3C	on-policy	9.0	16042.70
PPO	on-policy	8.4	33766.36
DDPG	off-policy	3.0	13879.76
TD3	off-policy	3.0	20145.44
SAC	off-policy	3.0	21238.81

Table 6.3: Best model results. Results for reinforcement learning algorithm are obtained by running 10 trial that each contain 500 random starting point

Algorithms	Type	Mean Success Rate (%)	Episode Length	One-Step Execution Duration (milliseconds)
TRPO	on-policy	95.91 $\pm$ 0.47	98.06 $\pm$ 5.92	1.049
A3C	on-policy	75.35 $\pm$ 0.84	120.12 $\pm$ 18.15	1.023
PPO	on-policy	86.14 $\pm$ 0.76	106.48 $\pm$ 9.63	1.050
DDPG	off-policy	84.80 $\pm$ 0.98	143.07 $\pm$ 14.67	0.752
TD3	off-policy	98.70 $\pm$ 0.25	141.05 $\pm$ 19.36	0.696
SAC	off-policy	92.50 $\pm$ 0.68	113.38 $\pm$ 10.53	0.723
Classical Controller	conventional	100.00 $\pm$ 0.00	163.00 $\pm$ 0.00	0.200

forcement learning performs better in the starting positions they are trained at but sometimes fails in the unknown region. Note that, creating a trajectory costs 11.352 milliseconds which is not tabulated since we have created the trajectories offline.

6.4 Analyzing the Effects of Different Factors

In this section effects of adding DoF to the aerial manipulator, increasing and randomizing the goal position, and adding instructive to the training are examined for the deep learning algorithms.

6.4.1 Effects of Goal and Start Position on the Learning-based Approaches

We have conducted this experiment to see the effects of the box position goal on the learning process. In order to have a random destination position for the box, we had

to change the state space of the simulation environment where 2 new states are added. One of them being the destination position in x direction p_{des_x} and the other one being the destination position in the y -direction p_{des_y} , both of which are in inertial frame. So mathematically the state space became:

$$s_t = [p_{b_x}, p_{b_y}, p_{p_x}, p_{p_y}, p_{g_x}, p_{g_y}, p_{des_x}, p_{des_y}, \psi, \theta, \gamma, \beta, v_{p_x}, v_{p_y}, \dot{\theta}, G]. \quad (6.1)$$

For the first experiment, the random region of the end goal is defined as:

$$p_{des_x} \sim U(-0.1, 0.1), \quad (6.2)$$

$$p_{des_y} = -0.25, \quad (6.3)$$

which is defined as small random destination goal in Figure 6.10 and the second experiment is defined as:

$$p_{des_x} \sim U(-0.3, 0.3), \quad (6.4)$$

$$p_{des_y} \sim U(-0.3, 0.3), \quad (6.5)$$

which is defined as the big random destination goal in Figure 6.10. Note that for the big random goal experiment to achieve the same results as the other ones, the architecture had to be doubled. Without this change, there was no convergence so we can safely say that randomizing the goal destination and increasing the region of the goal increases the problem complexity.

Moreover, the effects of starting position on off-policy and on-policy algorithms are also examined. we have selected the best models that result in the best reward and success rate and tried them on 500 different starting points to see the effects of the starting positions which can be seen in Figure 6.11. From these plots, it can be seen that there is no obvious bias toward any region for success or fail and TRPO achieves the best success rate. The details of the success rates of the best models that achieve the best rewards for 100 random episodes are given in Section 6.3.

And just like for the on policy algorithms, best models for the off policies are taken and tried on 500 starting points to see the effects of the starting point which can be seen in Figure 6.12 and the same as the on policy case there seem to be no obvious cause of bias.

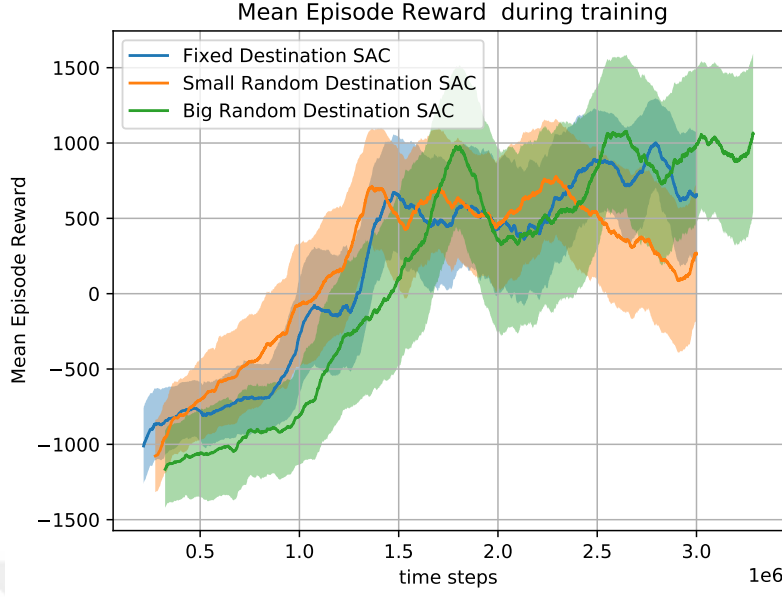


Figure 6.10: Comparison of mean reward of the SAC algorithm for different destination positions of the box.

6.4.2 Effects of DoF on the Learning-based Approaches

In addition to the comparison of algorithms for the task at hand, we have also conducted experiments to see the effects of increasing the DoF of the manipulator on the learning algorithms. For these tests, the SAC algorithm is chosen since its convergence to a reasonable reward is faster than the other algorithms and it is much more stable than the other algorithms.

We have created two additional manipulators where the first one can only move the grippers and the second one has one more joint at the center of the mass of the manipulator to have more freedom in movement. For this new 2-Dof and 4-Dof manipulator, one less and one more action of joint speed command is subtracted and added respectively. So, for the 2-Dof case, the action space is:

$$a_t = [f_l, f_r, v_{j_{gripper}}], \quad (6.6)$$

whereas, for the 4-Dof manipulator, the action space is :

$$a_t = [f_l, f_r, v_{j_{arm}}, v_{j_{arm_2}}, v_{j_{gripper}}], \quad (6.7)$$

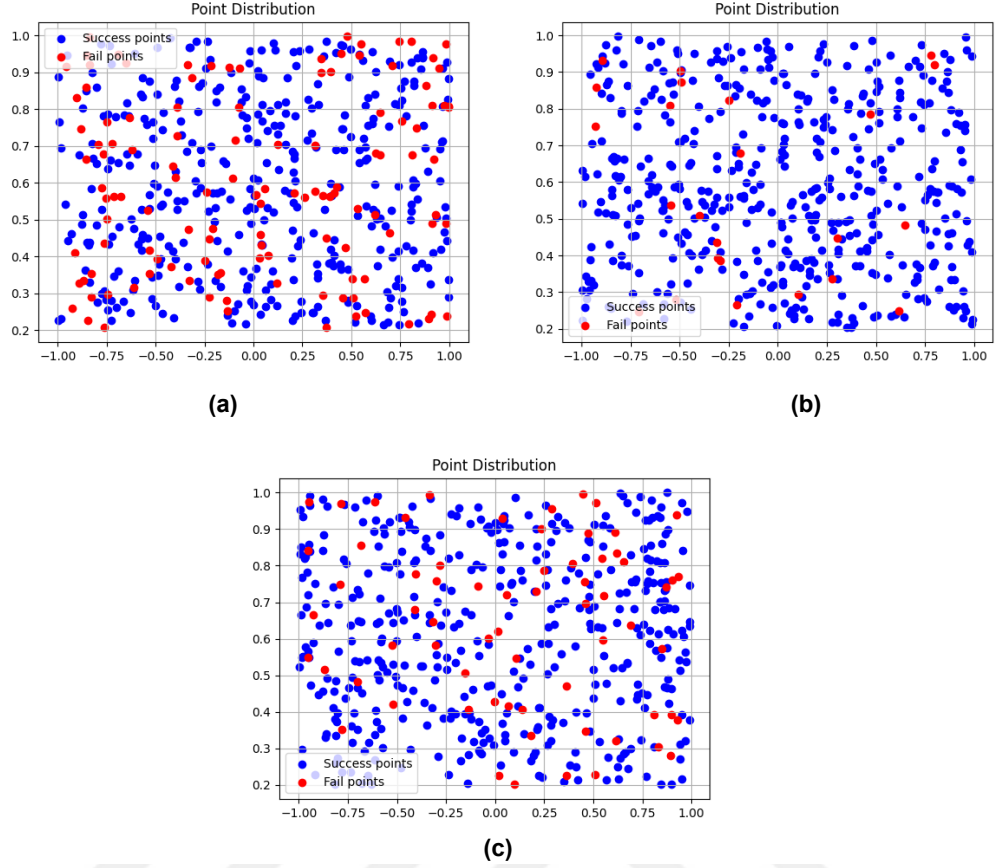


Figure 6.11: Point Distribution of the best models obtained from the algorithms (a) A2C, (b) TRPO, (c) PPO. Blue dots represents the successful starting points and the red dots represents the failed starting points.

where $v_{j_{arm_2}}$ is the additional revolute joint speed command. In all aerial manipulators, the network architecture is kept the same for a fair comparison.

The mean episode reward during training can be seen in the left figure of the Figure 6.13 and since the action space is decreased as expected the convergence of the 2-DoF is faster but from the right figure of the Figure 6.13, it can be seen that although all of the manipulators achieve a similar episode length, 3-DoF manipulator achieves the fastest mean episodes. So, although the problem becomes easier as we decrease the complexity, the task becomes harder (slower to solve) due to having less freedom to move.

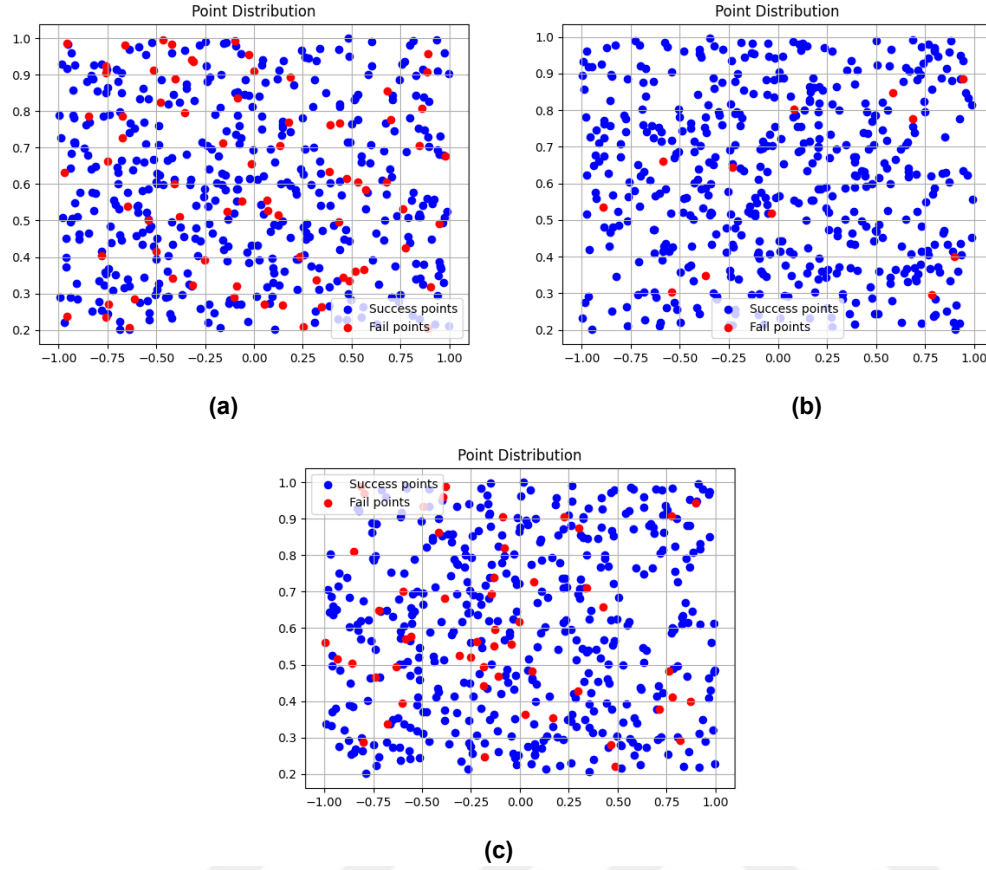


Figure 6.12: Point Distribution of the best models obtained from the algorithms (a) DDPG, (b) TD3, (c) SAC. Blue dots represents the successful starting points and the red dots represents the failed starting points.

6.4.3 Effects of Instructive State on the Learning-based Approaches

Last but not least, we show the benefit of adding an informative state to the training of the learning approaches in this experiment. The informative state for the aerial manipulator pick and place operation is that the aerial manipulator starts with the box already gripped to reduce the exploration process. In this experiment the informative state occurs with a probability of 0.1 for each episode and for the rest the training process starts normally. As can be seen in Figure 6.14 adding information state enables the algorithm to converge faster with less exploration.

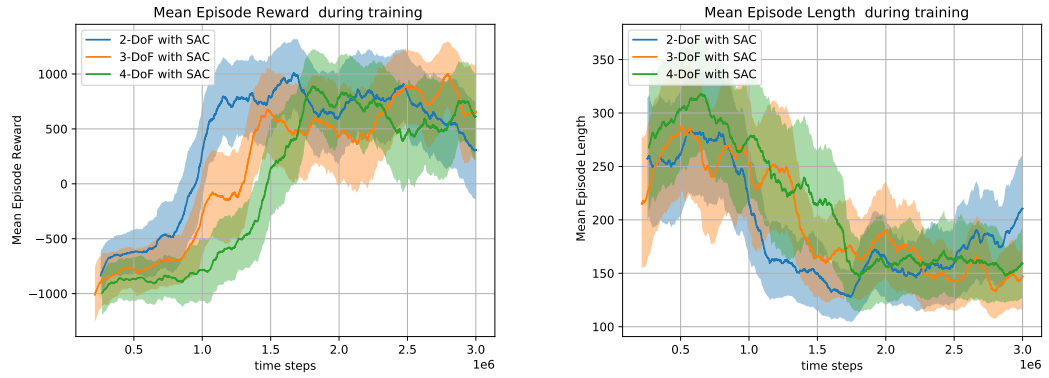


Figure 6.13: Comparison of mean reward (left) and mean length (right) of the SAC algorithm for different degree of freedom manipulators.

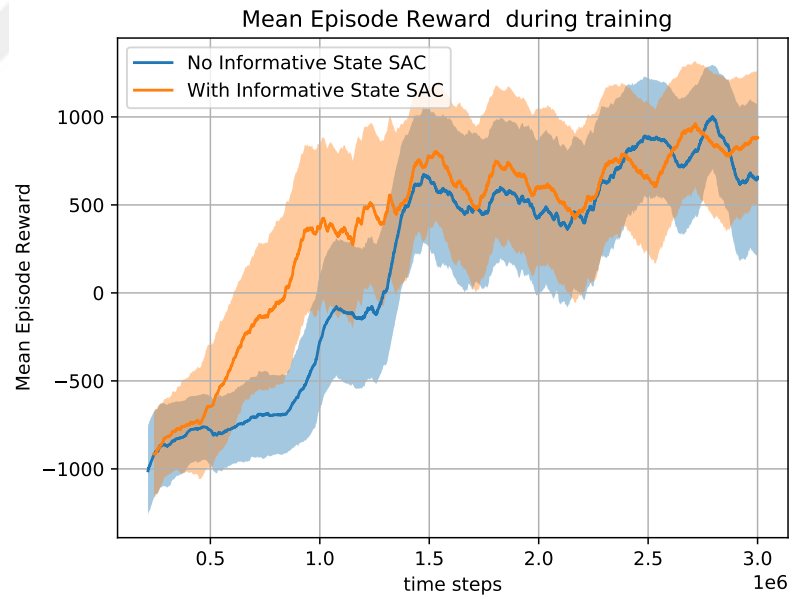


Figure 6.14: Comparison of mean reward of the SAC algorithm for adding informative state to the training.



CHAPTER 7

CONCLUSION

In this thesis, the control of a simplified 2D aerial manipulator for a pick and place operation is considered. Two types of controller designs were examined. The first controller considered is the classical controller, which is a cascaded approach similar to the quad-rotor controllers. This controller obtains position, velocity and angle commands from a trajectory generation algorithm, called minimum snap trajectory, and creates the required forces and torque to follow this trajectory. The second set of controllers are learning-based approaches to solve the control problem in an end-to-end fashion. For the learning approaches, the state-of-the-art methods, namely TRPO, A2C, PPO, DDPG, TD3, SAC, were examined. These methods are trained to learn a non-linear controller with a trial & error fashion where the algorithms explore the possible action to take in a state and then exploit what they learned to obtain the maximum return from the environment.

In the classical approach, minimum snap trajectories are generated offline with a QP solver, in order to use the same trajectories multiple times, and fed into the conventional controller that is in control of position and attitude. For the optimization of the control parameters, two optimization algorithms are used, namely, PSO and Nelder-Mead. PSO is used for global tuning whereas Nelder-Mead is used for fine-tuning the controller parameters. For the cost function of both of these optimization methods, the reward function of the learning approaches is used for a fair comparison between the classical approach and learning approaches.

In deep reinforcement learning approaches, a simulation environment is used for training the agents. Just with a reward signal feedback from the environment, all algorithms learned how to perform the pick and place operation of an object with an

aerial manipulator with a non-linear neural network controller architecture. The network maps the states of the environment to the required actions to perform the task at hand.

Our experiments show that with deep reinforcement learning, it is possible to perform pick and place operations with aerial manipulators. In addition to this, in regions where learning approaches finish the task successfully, they outperform the conventional controller approach in terms of speed.

Though only a somewhat simple reward function is required for the training of the deep reinforcement learning approaches and they give valuable results in local minimum over parameters of policy they use, the action outputs of these algorithms differ greatly if there is a small change in state and sometimes even completely fail in some cases. Moreover, the stability of these algorithms is a bit complicated. This not the case for the classical controller case though. In the classical controller approach, the stability of the controller is mathematically proven and the controller for the given task does not fail if there is a small change in state. However, the classical controller requires robot dynamics to be known and this can make the controller design and implementation harder and some assumptions made to ease the design may make the solution sub-optimal which is not the case for the learning approaches.

We also analyzed the effects of several crucial factors on the controllers. For example, the effects of increasing the degrees of freedom of the manipulator on learning approaches. Though decreasing the degree of freedom results in faster convergence, the overall task becomes harder to complete and the agent requires more time to complete the task. Adding more degrees of freedom, on the other hand, makes the training harder which requires more work that may not also be beneficial for the task at hand. In addition to this, the effects of randomizing the goal position are also considered for the learning approaches. It is shown that increasing the goal region, increases the complexity which can make the problem unsolvable with the given solution. Finally, we have also shown that changing the initial state to a more informative state like starting the aerial manipulator with the box already gripped, increases the convergence rate for the learning approaches.

Though this thesis can be considered as a starting point for solving the pick and place

operation with deep reinforcement learning since the simulation environment is in 2D, it is not possible to show the results in the real world. Since the learning algorithms are fairly small, it is possible to use them in the real world though. In addition to this, as we discussed, as the region of the training increase for the learning algorithms, the complexity of the problem increases such that it becomes unsolvable. To solve this, methods like HRL (Hierarchical Reinforcement Learning) can be used to divide the problem into parts.





REFERENCES

- [1] G. Heredia, A. Jimenez-Cano, I. Sanchez, D. Llorente, V. Vega, J. Braga, J. Acosta, and A. Ollero, “Control of a multirotor outdoor aerial manipulator,” in *2014 IEEE/RSJ international conference on intelligent robots and systems*, pp. 3417–3422, IEEE, 2014.
- [2] J. Thomas, G. Loianno, J. Polin, K. Sreenath, and V. Kumar, “Toward autonomous avian-inspired grasping for micro aerial vehicles,” *Bioinspiration & biomimetics*, vol. 9, no. 2, p. 025010, 2014.
- [3] F. Ruggiero, V. Lippiello, and A. Ollero, “Aerial manipulation: A literature review,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1957–1964, 2018.
- [4] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, “Data-efficient deep reinforcement learning for dexterous manipulation,” *arXiv preprint arXiv:1704.03073*, 2017.
- [5] C. Wang, Q. Zhang, Q. Tian, S. Li, X. Wang, D. Lane, Y. Petillot, and S. Wang, “Learning mobile manipulation through deep reinforcement learning,” *Sensors*, vol. 20, no. 3, p. 939, 2020.
- [6] H. B. Khamseh, F. Janabi-Sharifi, and A. Abdessameud, “Aerial manipulation—a literature survey,” *Robotics and Autonomous Systems*, vol. 107, pp. 221–235, 2018.
- [7] D. Mellinger, M. Shomin, and V. Kumar, “Control of quadrotors for robust perching and landing,” in *Proceedings of the International Powered Lift Conference*, pp. 205–225, 2010.
- [8] H. A. Almurib, P. T. Nathan, and T. N. Kumar, “Control and path planning of

- quadrotor aerial vehicles for search and rescue,” in *SICE Annual Conference 2011*, pp. 700–705, IEEE, 2011.
- [9] K. Sreenath, T. Lee, and V. Kumar, “Geometric control and differential flatness of a quadrotor uav with a cable-suspended load,” in *52nd IEEE Conference on Decision and Control*, pp. 2269–2274, IEEE, 2013.
- [10] H. Li, B. Wang, L. Liu, G. Tian, T. Zheng, and J. Zhang, “The design and application of smartcopter: An unmanned helicopter based robot for transmission line inspection,” in *2013 Chinese Automation Congress*, pp. 697–702, IEEE, 2013.
- [11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [12] S. Fujimoto, H. Van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” *arXiv preprint arXiv:1802.09477*, 2018.
- [13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *arXiv preprint arXiv:1801.01290*, 2018.
- [14] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International conference on machine learning*, pp. 1889–1897, 2015.
- [15] 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*, pp. 1928–1937, 2016.
- [16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [17] H. I. Ugurlu, “A comparative study of learning based control policies and conventional controllers on 2d bi-rotor platform with tail assistance,” Master’s thesis, Middle East Technical University, 2019.

- [18] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation*, pp. 2520–2525, IEEE, 2011.
- [19] M.-D. Hua, T. Hamel, P. Morin, and C. Samson, "Introduction to feedback control of underactuated vtolvehicles: A review of basic control design ideas and principles," *IEEE Control systems magazine*, vol. 33, no. 1, pp. 61–75, 2013.
- [20] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, p. 65, 2010.
- [21] N. Dadkhah and B. Mettler, "Survey of motion planning literature in the presence of uncertainty: Considerations for uav guidance," *Journal of Intelligent & Robotic Systems*, vol. 65, no. 1-4, pp. 233–246, 2012.
- [22] A. Bicchi and V. Kumar, "Robotic grasping and contact: A review," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1, pp. 348–353, IEEE, 2000.
- [23] J. Fink, N. Michael, S. Kim, and V. Kumar, "Planning and control for cooperative manipulation and transportation with aerial robots," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 324–334, 2011.
- [24] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of cubic structures with quadrotor teams," *Proc. Robotics: Science & Systems VII*, 2011.
- [25] M. Orsag, C. M. Korpela, S. Bogdan, and P. Y. Oh, "Hybrid adaptive control for aerial manipulation," *Journal of intelligent & robotic systems*, vol. 73, no. 1-4, pp. 693–707, 2014.
- [26] S. Kim, S. Choi, and H. J. Kim, "Aerial manipulation using a quadrotor with a two dof robotic arm," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4990–4995, IEEE, 2013.
- [27] C. Korpela, M. Orsag, and P. Oh, "Towards valve turning using a dual-arm aerial manipulator," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3411–3416, IEEE, 2014.

- [28] H. B. Khamseh, F. Janabi-Sharifi, and A. Abdessameud, “Aerial manipulation—a literature survey,” *Robotics and Autonomous Systems*, vol. 107, pp. 221–235, 2018.
- [29] V. Lippiello and F. Ruggiero, “Cartesian impedance control of a uav with a robotic arm,” *IFAC Proceedings Volumes*, vol. 45, no. 22, pp. 704–709, 2012.
- [30] S. Kannan, M. A. Olivares-Mendez, and H. Voos, “Modeling and control of aerial manipulation vehicle with visual sensor,” *IFAC Proceedings Volumes*, vol. 46, no. 30, pp. 303–309, 2013.
- [31] G. Antonelli and E. Cataldi, “Adaptive control of arm-equipped quadrotors. theory and simulations,” in *22nd Mediterranean Conference on Control and Automation*, pp. 1446–1451, IEEE, 2014.
- [32] B. Yang, Y. He, J. Han, and G. Liu, “Rotor-flying manipulator: modeling, analysis, and control,” *Mathematical Problems in Engineering*, vol. 2014, 2014.
- [33] S. Kim, S. Choi, and H. J. Kim, “Aerial manipulation using a quadrotor with a two dof robotic arm,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4990–4995, IEEE, 2013.
- [34] G. Antonelli and E. Cataldi, “Adaptive control of arm-equipped quadrotors. theory and simulations,” in *22nd Mediterranean Conference on Control and Automation*, pp. 1446–1451, IEEE, 2014.
- [35] F. Ruggiero, M. A. Trujillo, R. Cano, H. Ascorbe, A. Viguria, C. Pérez, V. Lippiello, A. Ollero, and B. Siciliano, “A multilayer control for multirotor uavs equipped with a servo robot arm,” in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 4014–4020, IEEE, 2015.
- [36] A. E. Jimenez-Cano, J. Martin, G. Heredia, A. Ollero, and R. Cano, “Control of an aerial robot with multi-link arm for assembly tasks,” in *2013 IEEE International Conference on Robotics and Automation*, pp. 4916–4921, IEEE, 2013.
- [37] J. Escareno, G. Flores, M. Rakotondrabe, H. Romero, R. Lozano, and E. Rubio, “Task-based control of a multirotor miniature aerial vehicle having an onboard manipulator,” in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 857–863, IEEE, 2014.

- [38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [39] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Advances in neural information processing systems*, pp. 1057–1063, 2000.
- [40] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” 2014.
- [41] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396, IEEE, 2017.
- [42] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, “Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors,” *arXiv preprint arXiv:1903.04628*, 2019.
- [43] O. Kilinc, Y. Hu, and G. Montana, “Reinforcement learning for robotic manipulation using simulated locomotion demonstrations,” *arXiv preprint arXiv:1910.07294*, 2019.
- [44] W. Koch, R. Mancuso, R. West, and A. Bestavros, “Reinforcement learning for uav attitude control,” *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019.
- [45] S. Mysore, B. Mabsout, K. Saenko, and R. Mancuso, “How to train your quadrotor: A framework for consistently smooth and responsive flight control via reinforcement learning,” *arXiv preprint arXiv:2012.06656*, 2020.
- [46] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a quadrotor with reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

- [47] C.-H. Pi, K.-C. Hu, S. Cheng, and I.-C. Wu, “Low-level autonomous control and tracking of quadrotor using reinforcement learning,” *Control Engineering Practice*, vol. 95, p. 104222, 2020.
- [48] C. Li, F. Xia, R. Martín-Martín, and S. Savarese, “Hrl4in: Hierarchical reinforcement learning for interactive navigation with mobile manipulators,” in *Conference on Robot Learning*, pp. 603–616, PMLR, 2020.
- [49] T. Welschhold, C. Dornhege, and W. Burgard, “Learning mobile manipulation actions from human demonstrations,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3196–3201, IEEE, 2017.
- [50] A. Iriondo, E. Lazkano, L. Susperregi, J. Urain, A. Fernandez, and J. Molina, “Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning,” *Applied Sciences*, vol. 9, no. 2, p. 348, 2019.
- [51] J. Escareno, M. Rakotondrabe, G. Flores, and R. Lozano, “Rotorcraft mav having an onboard manipulator: Longitudinal modeling and robust control,” in *2013 European Control Conference (ECC)*, pp. 3258–3263, IEEE, 2013.
- [52] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [53] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*. Princeton university press, 2015.
- [54] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [55] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An introduction to deep reinforcement learning,” *arXiv preprint arXiv:1811.12560*, 2018.
- [56] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, “Stable baselines.” <https://github.com/hill-a/stable-baselines>, 2018.

- [57] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, “Openai baselines.” <https://github.com/openai/baselines>, 2017.
- [58] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, “Learning to reinforcement learn,” *arXiv preprint arXiv:1611.05763*, 2016.
- [59] G. E. Uhlenbeck and L. S. Ornstein, “On the theory of the brownian motion,” *Physical review*, vol. 36, no. 5, p. 823, 1930.
- [60] E. CATTO, “Box2d: A 2d physics engine for games,” 2011.
- [61] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [62] R. Eberhart and J. Kennedy, “Particle swarm optimization,” in *Proceedings of the IEEE international conference on neural networks*, vol. 4, pp. 1942–1948, Citeseer, 1995.
- [63] F. Gao and L. Han, “Implementing the nelder-mead simplex algorithm with adaptive parameters,” *Computational Optimization and Applications*, vol. 51, no. 1, pp. 259–277, 2012.
- [64] MATLAB, 9.7.0.1190202 (R2019b). Natick, Massachusetts: The MathWorks Inc., 2018.