A COMPARATIVE STUDY OF LEARNING BASED CONTROL POLICIES AND CONVENTIONAL CONTROLLERS ON 2D BI-ROTOR PLATFORM WITH TAIL ASSISTANCE

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

BY

HALIL İBRAHIM UĞURLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

A COMPARATIVE STUDY OF LEARNING BASED CONTROL POLICIES AND CONVENTIONAL CONTROLLERS ON 2D BI-ROTOR PLATFORM WITH TAIL ASSISTANCE

submitted by **HALIL İBRAHIM UĞURLU** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

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

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Afşar Saranlı
Supervisor, **Electrical and Electronics Engineering, METU** _____

Assoc. Prof. Dr. Sinan Kalkan
Co-supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Afşar Saranlı
Electrical and Electronics Engineering, METU _____

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

Assist. Prof. Dr. Mustafa Mert Ankaralı
Electrical and Electronics 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: Halil İbrahim Uğurlu

Signature :

ABSTRACT

A COMPARATIVE STUDY OF LEARNING BASED CONTROL POLICIES AND CONVENTIONAL CONTROLLERS ON 2D BI-ROTOR PLATFORM WITH TAIL ASSISTANCE

Uğurlu, Halil İbrahim
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Afşar Saranlı
Co-Supervisor: Assoc. Prof. Dr. Sinan Kalkan

September 2019, 59 pages

With the developing technology, multi-rotor platforms have become widespread and their control has become an important problem. In this thesis, we analyze physical extensions and control approaches for better control of rotor platforms. The first main contribution of the thesis is whether a tail-appendage that is attached under a multi-rotor platform can improve the multi-rotor's performance. Moreover, we used conventional control approaches as well as Deep Reinforcement Learning to learn a policy for controlling rotor platforms with or without tail appendage. To obtain better training and testing performance with Deep Reinforcement Learning, we adopted a curricular learning approach, where the difficulty of training samples is gradually increased. For the experiments, a two-dimensional simulation environment is developed to simulate a bi-rotor flying system, the counterpart of quad-rotors in three-dimensions. Both control strategies are rigorously analyzed for controlling the platform with and without tail appendage in this simulation environment.

Keywords: Deep Reinforcement Learning, multi-rotor UAVs, Artificial Neural Networks

ÖZ

ÖĞRENME TEMELLİ KONTROLCÜLER İLE GELENEKSEL KONTROLCÜLERİN İKİ BOYUTTA KUYRUKLA DESTEKLENMİŞ İKİ ROTORLU UÇAN ROBOTİK PLATFORM ÜZERİNDE KARŞILAŞTIRMALI ÇALIŞMASI

Uğurlu, Halil İbrahim
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Afşar Saranlı
Ortak Tez Yöneticisi: Doç. Dr. Sinan Kalkan

Eylül 2019 , 59 sayfa

Gelişen teknoloji ile birlikte çok rotorlu platformlar yaygınlaştı ve kontrolleri önemli bir problem haline geldi. Bu tezde, rotor platformlarının daha iyi kontrolü için fiziksel uzantıları ve kontrol yaklaşımlarını analiz ediyoruz. Tezin ilk ana katkısı, çok rotorlu bir platformun altına yerleştirilmiş kuyruk eklentisinin çoklu rotorun performansını arttırıp arttırmayacağıdır. Ayrıca, kuyruk eklentisi olan veya olmayan rotor platformalarını kontrol etmek için geleneksel kontrol yaklaşımlarının yanı sıra Derin Pekiştirmeli Öğrenme'yi de kullandık. Derin Pekiştirmeli Öğrenme ile daha iyi eğitim ve test performansı elde etmek için, eğitim örneklerinin zorluğunun kademeli olarak arttığı bir müfredatla öğrenme yaklaşımı kullandık. Deneyler için, üç boyutlu bir dört-rotorun iki boyuttaki karşılığı olan iki-rotorlu uçan sistemi için bir benzetim ortamı geliştirdik. Her iki kontrol stratejisini, bu simülasyon ortamında kuyruk eklemesi olan ve olmayan platformun kontrolü için titizlikle analiz ettik.

Anahtar Kelimeler: Derin Pekiştirmeli Öğrenme, çoklu-rotorlu İHAlar, Yapay Sinir Ağları

To my dear family...

"Plans are worthless, but planning is everything"

Dwight D. Eisenhower

ACKNOWLEDGMENTS

First and foremost, I would first like to express my sincere appreciation and gratitude to my supervisors Assoc. Prof. Dr. Afşar Saranlı and Assoc. Prof. Dr. Sinan Kalkan for their continuous support, criticism and invaluable guidance throughout my thesis study. For their comments and criticism, I would also like to thank the examining committee members; Prof. Dr. İlkay Ulusoy, Assist. Prof. Dr. M. Mert Ankaralı, and Assoc. Prof. Dr. Mehmet Serdar Güzel.

I would like to express my gratitude to the mates from ATLAS Interdisciplinary Robotics Laboratory. Osman Kaan Karagöz is the one who sits on the next table. Ferhat Gölbol, Cem Önem, Başer Kandehir, Sinan Şahin Candan, Jeanpierre Demir and Görkem Seçer made the hours I spend in laboratory enjoyable. Lütfullah Tomak and Nurullah Gülmüş are also my roommates from the department.

I would like to mention my friends, Furkan Karakaya, Hasan Burhan Beytur, Yusuf Candan, Salih Gedük for our friendly conversations on deep or small issues through my Ms. journey.

I am thankful to METU-BAP, coordinator-ship of scientific research projects, for their financial support under GAP-312-2018-2705.

I am thankful to TÜBİTAK, the National Scientific and Technological Research Council of Turkey, for granting me their M.S. studies scholarship.

Finally, but forever I owe my loving thanks to my family, my loving mother Emine, my dear father Davut, my brothers Muhammed and Musab, my sister Zeyneb but especially my beloved wife Hacer Banu for their undying love, support, and encouragement.

"And he made it a word enduring among his posterity; haply so they would return."

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
UAV	Unmanned Aerial Vehicle
MDP	Markov Decision Processes
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
TRPO	Trust Region Policy Optimization
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
MLP	Multilayer Perceptron
KL	Kullback–Leibler
PD	Proportional-Derivative
CoM	Center of Mass
p2p	Point-to-point

LIST OF VARIABLES

M	Mass of bi-rotor body
m	Mass of tail
I	Inertia of bi-rotor body
I_{tail}	Inertia of tail
g	Gravitational acceleration
$\mathbf{x}$	State vector of bi-rotor system
p_x	Position of bi-rotor in x-direction
p_y	Position of bi-rotor in y-direction
ϑ	Orientation of bi-rotor
v_x	Velocity of bi-rotor in x-direction
v_y	Velocity of bi-rotor in y-direction
ω	Angular velocity of bi-rotor
φ	Orientation of tail with respect to bi-rotor body
$\mathbf{u}$	Input vector of bi-rotor system
f_l	Thrust of left rotor
f_r	Thrust of right rotor
τ_{tail}	Torque applied to tail
τ	Equivalent torque of left and right rotor thrusts
f	Sum of left and right rotor thrusts
s	State of an MDP
a	Action of an MDP
r	Reward of an MDP

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

Quad-rotor UAVs are one of the most widely studied vehicles in aerial robotics and multi-rotor platforms [3, 4, 5]. Generally, they work on four propellers, which can apply forces perpendicular to robot body only. They are studied in variable scales from human carrying level [6], to micro level such as Bitcraze's Crazyflie quad-rotor UAV [1] shown in Figure 1.1.

Figure 1.1: A crazyflie 2.0 quad-rotor UAV [1]

A *bi-rotor* is a conceptual equivalent of quad-rotor in two-dimensional space. In this work, we focused on the control of a bi-rotor. Detailed dynamics of the bi-rotor system will be given in Chapter 3. In Figure 1.2, Quanser's 3-DOF helicopter setup is shown [2]. This is an example of bi-rotor notion implemented as a real-world setup by decremented degrees of freedom. Similar to a quad-rotor UAV platform, a bi-rotor has two parallel rotors placed at both ends of a stick like body.

Figure 1.2: Quanser's 3-DOF bi-rotor helicopter setup [2]

A multi-rotor system has a non-linear, non-holonomic, underactuated structure. Therefore, their control is challenging [7]. As an intuitive example of control complexity, because of its rotor directions, in order to move in some horizontal direction, a multi-rotor needs to tilt in that direction first and then recover its orientation without falling down. There are controllers developed by traditional control-theoretic approaches [8, 9, 7]. However, they necessitate system-identification and parameter tuning. To avoid using such controllers, some deep learning based control policies have also been proposed [10, 11, 12, 13]. In particular, along with advances in Deep Reinforcement Learning methods starting from Deep Q-Networks [14], those are employed to control dynamical robotic systems from legged [15] to aerial [13] systems. Especially, policy-gradient based methods are well suited for robotics problems because of their capability to handle high-dimensional state spaces.

1.2 Contributions

In this thesis, a comparative investigation is conducted between control theoretic approaches and learning based methods on a bi-rotor platform. We design controllers to locate bi-rotor platform at a point in space and drive it to that point. The control theoretic approach is called a *conventional controller* in this text and implemented with cascaded controllers for attitude and positional control of bi-rotor as done for quad-rotor control. Moreover, Proximal Policy Optimization (PPO) [16] is used as the learning-based method. PPO is a state-of-the-art Deep Reinforcement Learning

algorithm and shows its success in stabilizing a quad-rotor UAV [13].

We also equip the bi-rotor with a torque actuated under-hanging tail appendage. The conventional controller is extended to use the tail for attitude control. A control policy is also trained in this setting and the agent is observed how to use the tail appendage. Although tail-like appendages have been shown to help in land or aerial vehicles [17, 18, 19], we employ such a mechanism in a learning framework and study its effects.

Lastly, we apply a curricular approach which is a method used in Supervised Learning domains. Specifically, curriculum learning [20] notion is based on scheduling the learning process first with easy samples and making harder step by step. Accordingly, we train the agent first with an easy domain and broaden the domain by time.

1.3 The Outline of the Thesis

This thesis work organized as follows. In Chapter 2, current situation of the literature about the control of flying rotor platforms and tail assistance in Robotics. Chapter 3 gives the mathematical background about bi-rotor dynamics, control and Reinforcement Learning. Dynamics of tail appendage and design of simulation environment are explained in Chapter 4. In Chapter 5, control methodology for both conventional and learning based controllers are detailed. Experimental results are shared in Chapter 6. Finally, Chapter 7 concludes this thesis with future directions.

CHAPTER 2

RELATED WORK

2.1 Deep Reinforcement Learning

Advances in deep learning and novel neural network architectures made it possible to apply them to dynamic system control problems through *deep reinforcement learning* (DRL) and several methods proposed for DRL. At first, Mnih et al. [14] have demonstrated that one can train a network structure to learn to play early atari games simply by observing raw pixels and with rewards only defined on win-lose outcomes. The so called “Deep Q Networks" (DQN), being non-linear function approximators, are successfully demonstrated to learn *value functions* from such raw data.

An alternative method in the literature is to use a deep network structure to directly estimate the control policy rather than the value function and to train it using the gradients on the policy [21, 22]. Methods based on these so called *Policy Gradients*, were shown to provide better convergence properties, learn more sophisticated policies, and scale better to continuous action spaces. These may be considered earliest examples of continuous action spaces. As a more recent development, two networks are simultaneously trained [23] for the task: An *actor network* (i.e., a controller) that estimates an action given the current state of the environment, and a *critic network* that predicts the value function at a given state. These methods, called Actor-Critic Networks, are demonstrated to perform much better in a number of continuous control problems [23, 24].

In policy gradient methods, generally we do not want the updated policy be too different from the old policy concerning instabilities. To ensure this and improve convergence, Schulman et al. [25] presents Trust Region Policy Optimization (TRPO)

that limits the step sizes in a trust region by adding a KL divergence constraint on optimization. Later, they propose Proximal policy optimization (PPO) [16] which is an improved version of TRPO [25], and is currently the state of the art in policy gradient based Deep Reinforcement Learning algorithms. Differently, PPO uses a proportionality ratio between policies and clip them in a region instead of calculating KL divergence. Moreover, Schulman et al. show that clipping the objective function yields even better results.

2.2 Curriculum Learning

In addition to these methods, the data itself has also an impact on parameter improvements in neural networks. Bengio et al. observed that when the training data is presented to the network randomly, it takes longer for the network to converge in Supervised Learning [20]. Moreover, it is a well-known problem that deep learning is a non-convex optimization, and it suffers from local minima [26, 27]. They showed that starting the training procedure with simpler examples, and introducing gradually more complex ones as the network learns the easier subtasks improves the convergence speed and the quality of local minima. Curriculum based methods are not limited to supervised learning domain, and there are studies [28, 29, 30, 31] that combine curriculum learning notion into Deep Reinforcement Learning domain.

2.3 Multi-rotor Control

Controlling non-linear, underactuated, agile and high-speed platforms such as quadrotor UAVs is challenging, posing problems such as high state-space dimension, concerns of stability and robustness as well as considerations of response time[7, 10]. Hand-crafted, low-level controllers fall short in addressing all these challenges and therefore, learning-based approaches emerged in the literature as an alternative to classical control theory [10, 32, 33, 34, 11]. A recent review of these approaches is given in [7]. In a number of studies, different applications are considered: For example, in [32], a deep network is used to make a UAV follow a hand-drawn trajectory while [33, 34] trains a UAV to fly using its vision sensors through cluttered outdoor

environments. Also, there are differences in solution approaches: [33, 34] uses learning by imitation while [10] uses an iterative method for function fitting which receives supervision from a feedback-based controller. A reference controller is used in [11] to train the deep network to demonstrate its trajectory generalization capability.

With promising results in using DRL for robot control problems, similar approaches have been employed for quad-rotor UAV control. These include e.g. [35], which used a DQN to teach a UAV to avoid obstacles in indoor environments from a single image; [36], which combined model-predictive control with deep RL to obtain better performance and [12], which used an actor-critic network to teach a UAV to go to a close-by way-point. Their network maps the robot state to four actuator commands, hence requires no control theory background other than mathematical modeling of the quad-rotor. Their policy network controller is robust, even under very harsh initialization, the robot can stabilize itself successfully. To obtain the controller, Hwangbo et al. train two networks. Since their state vector is 18 dimensional and their action vector is 4 dimensional, input and output layers of the policy network have 18 and 4 neurons, respectively. They use two hidden layers, each with 64 neurons and tanh activation. Value network has the same architecture, except it has a single neuron in the output layer. They train value network using Huber loss [37], and policy network using natural gradient descent. As a more recent study, Molchanov et al. [13] employs PPO algorithm to learn a policy for multiple quad-rotors with different sizes.

CHAPTER 3

MATHEMATICAL BACKGROUND

In this chapter, the mathematical background for problem formulation will be given. First of all, in section 3.1 the dynamical model of bi-rotor platform is derived and control methodology is developed. Next, this control system is converted into a Markov Decision Process to apply Reinforcement Learning. In section 3.2, MDP formulation is derived from the dynamics of the system. Finally, in sections 3.3 and 3.4 Reinforcement Learning approach is introduced to find a solution to the MDP.

3.1 Bi-rotor Dynamics and Control

Bi-rotor is a planar equivalent of a quad-rotor. It has a linear body like a stick and two propellers at both ends as seen in Figure 3.1. All mass M and inertia I of the robot are assumed to be concentrated at the midpoint of the body, which is center of gravity.

The position of bi-rotor is defined by the location of the center of gravity in a 2D coordinate frame, given with (p_x, p_y). The orientation of bi-rotor is defined as the angle between the robot body and the positive x-axis, given with ϑ. There is a gravitational acceleration g at negative y direction affecting the bi-rotor body. The propellers apply forces f_l and f_r, perpendicular to the robot body.

Given these quantities, state vector of the bi-rotor system $\mathbf{x}$ is a 6x1 vector consisting of position, orientation, velocity and angular velocity of bi-rotor given as,

$$\mathbf{x} = \left[p_x, p_y, \vartheta, \dot{p}_x, \dot{p}_y, \dot{\vartheta}\right]^T, \tag{3.1}$$

that is, by definition of what the state is, includes enough information about the system

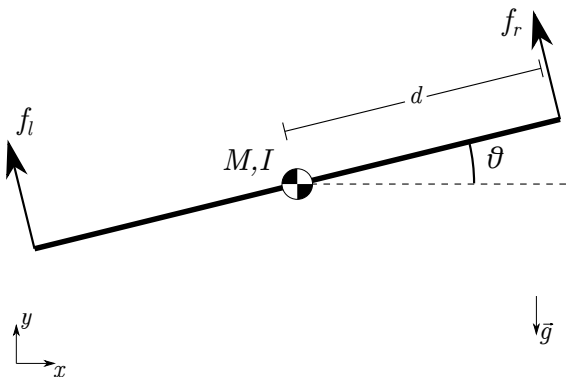

Figure 3.1: Our bi-rotor model

dynamics. This is also called as *Markovian Property* in Markov Decision Processes which will be refered in section 3.2. The input vector $\mathbf{u}$ consists of propeller forces given as,

$$\mathbf{u} = [f_l, f_r]^T \tag{3.2}$$

where $f_l, f_r \in [0, f_{max}]$ and f_{max} is the maximum thrust that one rotor can supply. A thrust-to-weight ratio is defined as,

$$r_{thrust2weight} = \frac{2f_{max}}{Mg}, \tag{3.3}$$

where M is the mass of bi-rotor and g is the gravitational acceleration.

We have velocities defined as,

$$\begin{aligned} v_x &= \dot{p}_x, \\ v_y &= \dot{p}_y, \\ \omega &= \dot{\vartheta}. \end{aligned} \tag{3.4}$$

By substituting velocities in Equation 3.4 into Equation 3.1, the state vector becomes,

$$\mathbf{x} = [p_x, p_y, \vartheta, v_x, v_y, \omega]^T. \tag{3.5}$$

Total forces acting on the body in the x and y directions are

$$\begin{aligned} f_x &= -(f_l + f_r)\sin\vartheta, \\ f_y &= (f_l + f_r)\cos\vartheta - Mg, \end{aligned} \tag{3.6}$$

and the total torque around the center of mass is,

$$\tau = (f_r - f_l) \cdot d, \tag{3.7}$$

where d is the half distance between left and right rotors.

According to Newton's second law, we have that,

$$\dot{v}_x = \frac{f_x}{M} = -\frac{f_l + f_r}{M} \sin\vartheta, \tag{3.8}$$

$$\dot{v}_y = \frac{f_y}{M} = \frac{f_l + f_r}{M} \cos\vartheta - g, \tag{3.9}$$

$$\dot{\omega} = \frac{\tau}{I} = \frac{(f_r - f_l) \cdot d}{I}. \tag{3.10}$$

Rearranging equations 3.8, 3.9 and 3.10 into state space representation form of,

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} + \mathbf{g}, \tag{3.11}$$

we get,

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\sin\vartheta/M & -\sin\vartheta/M \\ \cos\vartheta/M & \cos\vartheta/M \\ -d/I & d/I \end{bmatrix} \mathbf{u} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -g \\ 0 \end{bmatrix}. \tag{3.12}$$

Full state information is used to control this system. Figure 3.2 represents the block diagram of full state feedback. The *controller* takes the full state information and drives the system by supplying thrust inputs of the bi-rotor system accordingly. A reference input and error calculation is not shown in the figure since the main task to achieve in this work is to reach the zero state configuration. That is, positioning the bi-rotor at the center of the reference coordinate frame with zero velocity and parallel orientation to horizontal. So, the state fed to the controller is itself the error.

In this work, two kinds of controller structure are studied. The first one is inspired by current quad-rotor control methods. This controller structure is called as a *conventional controller* in this text and used as a baseline. Rest of this section describes this

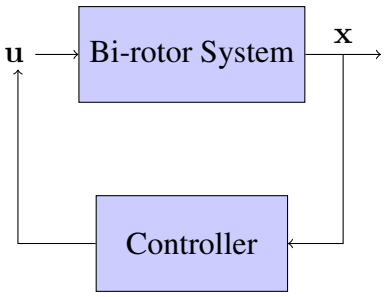

Figure 3.2: Block diagram of state feedback controller

structured controller. The second controller is represented as an artificial neural network that maps states to system inputs. The parameters of this controller network are learned with a Deep Reinforcement Learning method called Proximal Policy Optimization [16]. This controller is called a *learning-based controller* or a *control policy* which is the terminology used in Reinforcement Learning literature.

The conventional controller has a particular controller structure that separates angular and positional control of bi-rotor. The bi-rotor platform has two thrust inputs that are orthogonal to the platform surface and it is at equilibrium at zero orientation while thrust forces opposing the gravitational force. Adjusting the trusts by keeping them equal bi-rotor can be accelerated in vertical. From this equilibrium by deviating thrust forces, a net torque can be generated as can be seen from Equation 3.7. For horizontal movements, the system needs to break the zero orientation and can get a net force in the vertical direction.

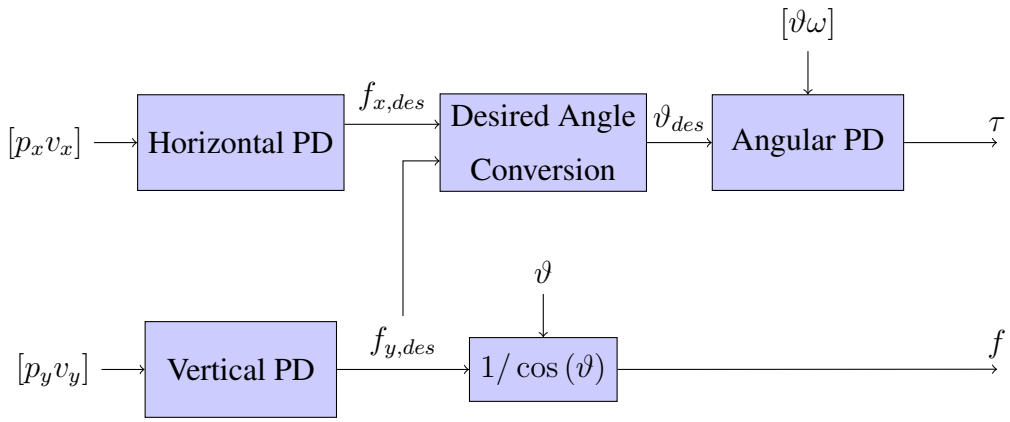

Figure 3.3: Block diagram of conventional controller

In Figure 3.3, the block diagram of the conventional controller is given. The complete diagram takes the terms of state vector as inputs and gives a torque and a force. The torque τ derived from rotor thrust difference as given in Equation 3.7 before. The force f is the total rotor thrust force applied to the center of mass of bi-rotor and those are defined as,

$$f = f_r + f_l. \tag{3.13}$$

Taking f_r and f_l from equations 3.7 and 3.13 we get,

$$\begin{aligned} f_r &= \frac{f}{2} + \frac{\tau}{2d}, \\ f_l &= \frac{f}{2} - \frac{\tau}{2d}, \end{aligned} \tag{3.14}$$

to calculate input vector $\mathbf{u}$ of the bi-rotor system from outputs of conventional controller.

There are five mathematical blocks represented in Figure 3.3 that are relating input terms to outputs mathematically. First, *Horizontal PD* block is a Proportional-Derivative (PD) controller in the horizontal direction. It outputs the required net force in horizontal as,

$$f_{x,des} = -(K_{p,x}p_x + K_{d,x}v_x), \tag{3.15}$$

where $K_{p,x}$ and $K_{d,x}$ are controller parameters. Similarly, *Vertical PD* block gives the required net force in vertical by,

$$f_{y,des} = Mg - (K_{p,y}p_y + K_{d,y}v_y), \tag{3.16}$$

where $K_{p,y}$ and $K_{d,y}$ are controller parameters. Third block calculates total rotor thrusts f such that its projection gives the desired vertical force:

$$f = f_{y,des}\frac{1}{\cos(\vartheta)}. \tag{3.17}$$

This equation provides that for higher ϑ values cosine becomes smaller and the total thrust command grows too much to supply from rotors. Since in this case, the bi-rotor cannot maintain the height level, it is necessary to keep its angle below a certain value. The orientational objective of bi-rotor is determined to provide the thrust forces

in both horizontal and vertical direction. That value is calculated in *Desired Angle Conversion* block as,

$$\vartheta_{des} = \text{clip}(\text{atan2}(f_{x,des}, f_{y,des}), \{\vartheta_{min}, \vartheta_{max}\}), \tag{3.18}$$

where the wanted value first calculated by 2-argument arctangent function, then clipped to keep it in a safe range where the clipping function is defined as,

$$\text{clip}(x, \{x_{min}, x_{max}\}) = \begin{cases} x_{min}, & x \leq x_{min} \\ x, & x_{min} < x < x_{max} \\ x_{max}, & x_{max} \leq x \end{cases}. \tag{3.19}$$

The clipping interval is a design parameter determined by considering thrust-to-weight ratio of bi-rotor given in equation 3.3 so that the bi-rotor can supply enough thrust to keep itself stable in vertical. Final block *Angular PD* is again a PD controller block that controls the attitude of bi-rotor as,

$$\tau = (K_{p,\vartheta}(\vartheta_{des} - \vartheta) + K_{d,\vartheta}(-\omega)), \tag{3.20}$$

where $K_{p,\vartheta}$ and $K_{d,\vartheta}$ are controller parameters.

3.2 Bi-rotor Representation as Markov Decision Process

Markov Decision Processes provide a mathematical framework for sequential decision-making problems including probabilistic state transitions. MDP is a straightforward framework for Reinforcement Learning problems [38] which we mention in next section. In this section, at first mathematical foundations of Markov Decision Processes are given and then the bi-rotor control problem is represented as a Markov Decision Process.

A Markov Decision Process is generally defined with 5-tuple: $(\mathcal{S}, \mathcal{A}, P, R, \rho_0)$. Here, $\mathcal{S}$ and $\mathcal{A}$ denotes the sets of all states and actions. The process has a state $s \in \mathcal{S}$ at discrete time step. For a state $s \in \mathcal{S}$ an action $a \in \mathcal{A}$ is taken, the process transitions into a new state $s' \in \mathcal{S}$ and gives a reward signal r. State-transition probabilities are given by P function:

$$P(s'|s, a) = Pr\{s_{t+1} = s'|s_t = s, a_t = a\}, \tag{3.21}$$

where, subscripts t and $t+1$ denotes the current and next time steps. Verbally, the next state of MDP is conditioned on the current state and action pair with a probability in $[0, 1]$. This is called the *Markov Property* that the future states of the process only depend on the current situation, and not to past.

The fourth element of MDP is a reward function R that supplies the reward signal depending on the current state-action pair and the next state:

$$r = R(s, a, s'). \tag{3.22}$$

A trajectory with sequential discrete time steps can be generated starting from an initial state $s_0 \in \mathcal{S}$:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots$$

where actions a_t's are selected by the *decision maker* and next state and reward is derived from equations 3.21 and 3.22. Initial state of the process is given by ρ_0 as a probability distribution;

$$\rho_0(s) = Pr\{s_0 = s\}. \tag{3.23}$$

In the problem of a Markov Decision Process, an *agent*, the *decision maker*, interacts with an *environment* that follows the properties of MDP. Action selection procedure of an agent is defined as a *policy*. Mathematically, the policy is represented as a probability distribution over actions given state:

$$\pi(a|s) = Pr\{a_t = a|s_t = s\}. \tag{3.24}$$

So far we give mathematical descriptions about Markov Decision Processes. Next, the bi-rotor system given in section 3.1 will be represented as a Markov Decision Process.

The states and actions of our MDP correspond to states and inputs of the bi-rotor

system. So, state and action sets are defined as the following;

$$\mathcal{S} = \{(p_x, p_y, \vartheta, v_x, v_y, \omega) | p_x, p_y, \vartheta, v_x, v_y, \omega \in \mathbb{R}\}, \tag{3.25}$$

$$\mathcal{A} = \{(f_l', f_r') | f_l', f_r' \in [-1, 1]\}, \tag{3.26}$$

where f_l' and f_r' are normalized thrust commands of propeller motors.

Hereby, the states and actions on bi-rotor is defined in continuous domain in contrast to vanilla MDP case. MDP formalisation can be extended to continuous state-action domain by defining state-transitions as probability distribution functions instead of probability mass functions. In our case, state-transitions happen according to the state space dynamic Equation 3.11 of the bi-rotor system with time discretization. That means, the input u given to the system is assumed to be constant for a small time step Δt and the next state of MDP is considered as the state of the bi-rotor system at that time instant as mathematical details will be explained in Section 4.1.1. Note that this model satisfies the Markov Property since the differential Equation 3.11 can be solved by the initial value of state $\mathbf{x}$ and independent from the past states.

Additionally, the Equation 3.11 can be transformed from continuous-time to discrete-time as a difference equation. For small enough time step, the solution of this difference equation becomes close to the solution of the differential equation.

Under these conditions the state-transitions do not include stochasticity and state transition function can be represented in a deterministic way by rewriting Equation 3.21 as,

$$s' = P(s, a). \tag{3.27}$$

Beyond these definitions, the reward function and the initial state distribution are design parameters defining the objective of the bi-rotor and details will be given in section 4.2.

3.3 Reinforcement Learning

Reinforcement Learning is simply defined as learning from interaction with an environment [38]. A general agent-environment interaction scheme in Reinforcement Learning is given in Figure 3.4. At a time step, the agent decides to take an action according to the current state observed from the environment. As a result of that action, the environment reaches a new state and produces a reward signal. The state transitions occur according to the dynamics of the environment, and the reward signal is the main objective that specifies the behavior of agent on environment. Briefly, a reinforcement learning agent is learning to get higher rewards continuously by interacting with the environment.

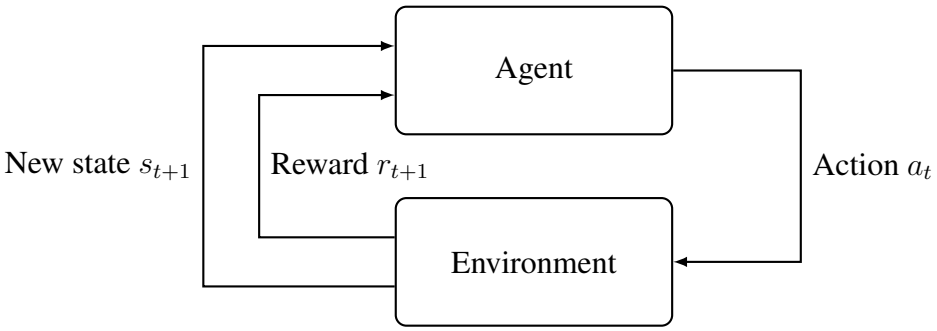

Figure 3.4: Agent-Environment interaction in Reinforcement Learning

The problem of reinforcement learning is formalized as control of Markov Decision Processes [38]. As explained before, the agent uses a policy that maps situations to actions and walks on the environment in the problem of MDP. This formalization implements sensation, action and the goal in a simple form and is very suitable for Reinforcement Learning problem in that sense.

At first, the agent has no prior information about how to act, instead, it should explore which actions return high reward. Another important challenge is that an action is not only affecting the current reward but affects future rewards through next states. Hereby, *trial-and-error* and *delayed reward* concepts are two main properties seen in Reinforcement Learning problems.

The reward signal is received one at a time step as given in Equation 3.22, and includes an instantaneous information. Aim of an agent is not gaining an immediate

high reward but looking after future rewards can be reached later before the termination of episode. Therefore, the *return* notion is defined to represent sum of future rewards collected through a trajectory as;

$$G_t = r_t + r_{t+1} + \cdots + r_T = \sum_{k=0}^{T-t} r_{t+k}, \tag{3.28}$$

where G_t is the return at time step t, equals to sum of all rewards of the sequence from time step t to termination time step T. Additionally, *discounted return* is defined as;

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T = \sum_{k=0}^{T-t} \gamma^k r_{t+k}, \tag{3.29}$$

where γ is the discount factor in $[0, 1]$ closed interval that provides a trade of between importance of short term or long term rewards.

A policy can be evaluated according to the expected value of discounted reward under that policy that is called a *value function* and defined as,

$$V^\pi(s) = \mathbb{E}[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | s_t = s, \pi], \tag{3.30}$$

where actions are derived by the policy π. The optimal value function is the maximum expected return under possible policies defined as;

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s), \tag{3.31}$$

where Π denotes the set of all policies. Similar to the value function, an *action-value function* is the expected return under an initial state-action pair and defined as,

$$Q^\pi(s, a) = \mathbb{E}[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi]. \tag{3.32}$$

The only difference of action-value function is that the first action is not derived by policy but specified as a parameter to the function. Similarly, the optimal action-value function is defined as;

$$Q^*(s, a) = \max_{\pi \in \Pi} Q^\pi(s, a). \tag{3.33}$$

Than, an *optimal policy* is the policy that yields the optimal value or action-value function, and the solution to the problem of Reinforcement Learning is that optimal policy. Additionally, an *advantage function* is difference between action-value and value functions and defined as,

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \tag{3.34}$$

This measure defines goodness of an action when compared to following the policy. These value function definitions are used in different algorithms solving Reinforcement Learning problems.

Instead of writing sum of rewards to the termination of episode in Equation 3.32, it can be derived in a recursive manner with one step forward. That is called a Bellman Equation [39] and defined for action-value function as,

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)(R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')), \tag{3.35}$$

where P and R are state transition and reward function defined in equations 3.21 and 3.22.

There are several methods proposed to solve the problem of Reinforcement Learning. They can be mainly separated as tabular methods and function approximation methods. In tabular methods value functions kept as tables and updated according to the Bellman Equation. For example in the q-learning algorithm [40], the action-value function is estimated on a table that contains all state-action pair. This table updated at every time step the agent moves and it is proved to converge to optimal action-value function [40]. The tabular methods are well suited for low-dimensional state and action spaces. However, function approximation methods fit for high-dimensional and even continuous state and action spaces. Those methods can also use a deep neural network for function approximation and this brings us to Deep Reinforcement Learning methods which will be covered in the next section.

3.4 Deep Reinforcement Learning

Deep Reinforcement Learning is a synthesis of Reinforcement Learning and Deep Learning [41]. It applies deep neural networks as function approximators for Reinforcement Learning methods. In particular, it represents the value or the policy functions as artificial neural networks to solve Reinforcement Learning problems. Even though there are a number of method proposed under Deep Reinforcement Learning, Proximal Policy Optimization (PPO) [16], a policy-gradient based algorithm, is employed in this thesis. In this section, the mathematical bases of value-based and

policy-gradient Deep Reinforcement Learning methods will be covered on the focus of PPO algorithm.

3.4.1 Value-based Methods

The goal of value-based algorithms is to construct a value function with neural networks. Later, a policy is derived from that value function. In this subsection, Deep Q-Network (DQN) [14] algorithm is covered particularly.

In DQN algorithm, experiences of an agent is stored in a replay buffer in the form of tuples $< s, a, r, s' >$ which are state, action, reward and next state of a transition on environment. This experience replay buffer allows to do gradient descent updates on value network with batches like supervised learning. The algorithm starts with an initial parameter distribution of action-value network $Q(s, a; \theta_0)$ where θ_0 is the initial parameters of network. At k^{th} update step target values of action-values are calculated as;

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{3.36}$$

where θ_k defines the parameters at that iteration. This target value is an estimation derived from Bellman Equation given in 3.35. Then, a square loss is defined as,

$$\mathbf{L}_{DQN} = (Q(s, a; \theta_k) - Y_k^Q)^2. \tag{3.37}$$

Using the gradient of that loss parameters of action-value network are updated using batches as;

$$\theta_{k+1} = \theta_k + \alpha(Y_k^Q - Q(s, a; \theta_k))\nabla_{\theta_k} Q(s, a; \theta_k), \tag{3.38}$$

where α is a scalar learning rate.

After convergence, the policy is derived on top of action-value network. In order to extract a policy from value function it is needed to have finite number of actions for the sake of computational complexity and algorithms designed for this. Thus, value-based methods do not fit to our bi-rotor control problem with continuous action choices in an interval.

DQN algorithm gives the essential idea behind value-based methods in Deep Reinforcement Learning. There are many variants of DQN proposed after that don't

covered in this text.

3.4.2 Policy Gradient Methods

Policy gradient methods optimize a parametric policy directly. From Equation 3.30, value of initial state under a stochastic policy can be derived as;

$$V^{\pi}(s_0) = \int_{\mathcal{S}} \rho^{\pi}(s) \int_{\mathcal{A}} \pi(s,a) R'(s,a) dads, \tag{3.39}$$

where $\rho^{\pi}(s)$ gives the state distribution with discount and defined as,

$$\rho^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t Pr\{s_t = s | s_0, \pi\} \tag{3.40}$$

Suppose the policy is parameterized with parameters ϖ, policy gradient theorem [21] gives,

$$\nabla_{\varpi} V^{\pi_{\varpi}}(s_0) = \int_{\mathcal{S}} \rho^{\pi_{\varpi}}(s) \int_{\mathcal{A}} \nabla_{\varpi} \pi_{\varpi}(s,a) Q^{\pi_{\varpi}}(s,a) dads. \tag{3.41}$$

Using the equivalence,

$$\nabla_{\varpi} \pi_{\varpi}(s,a) = \pi_{\varpi}(s,a) \frac{\nabla_{\varpi} \pi_{\varpi}(s,a)}{\pi_{\varpi}(s,a)} = \pi_{\varpi}(s,a) \nabla_{\varpi} \log(\pi_{\varpi}(s,a)), \tag{3.42}$$

the Equation 3.41 reduced to an expectation as,

$$\nabla_{\varpi} V^{\pi_{\varpi}}(s_0) = \mathbb{E}_{s \sim \rho^{\pi_{\varpi}}, a \sim \pi_{\varpi}} [\nabla_{\varpi} \log(\pi_{\varpi}(s,a)) Q^{\pi_{\varpi}}(s,a)]. \tag{3.43}$$

The policy can be updated step by step by estimating this gradient. This method uses action-value estimates that can be estimated by roll-outs on environment or an additional value estimator network can be constructed. The methods using both policy and value networks for this purpose are called actor-critic methods. The actor denotes the policy that take actions and the critic denotes the value network that criticizes the actor.

Furthermore, there are methods proposed to use Natural Gradients [42] instead of direct gradient. Modifying Natural Policy Gradients, Schulman et al. came up with Trust Region Policy Optimiation (TRPO) [25] that changes the policy in a controlled way with a constraint on Kullback Leibler (KL) divergence. Mathematically, their reformulated objective is defined using advantage function in Equation 3.34 as,

$$\max_{\Delta\varpi} \mathbb{E}_{s \sim \rho^{\pi_{\varpi}}, a \sim \pi_{\varpi}} [\frac{\pi_{\varpi + \Delta\varpi}(s,a)}{\pi_{\varpi}(s,a)} A^{\pi_{\varpi}}(s,a)], \tag{3.44}$$

subject to,

$$\mathbb{E}D_{KL}(\pi_{\varpi}(s,\cdot)||\pi_{\varpi+\Delta\varpi}(s,\cdot)) \leq \delta, \tag{3.45}$$

where δ is a hyperparameter and $\delta\varpi$ is the parameter update. By using Equation 3.45 as a penalty in Equation 3.44 we get,

$$\max_{\Delta\varpi} \mathbb{E}_{s\sim\rho^{\pi_{\varpi}},a\sim\pi_{\varpi}}[\frac{\pi_{\varpi+\Delta\varpi}(s,a)}{\pi_{\varpi}(s,a)}A^{\pi_{\varpi}}(s,a) - \beta D_{KL}(\pi_{\varpi}(s,\cdot)||\pi_{\varpi+\Delta\varpi}(s,\cdot))], \tag{3.46}$$

which is an unconstrained optimization that TRPO actually solves. Proximal Policy Optimization (PPO) [16] is a modification of TRPO algorithm. Defining the probability ratio as, $r_t(\Delta\varpi) = \frac{\pi_{\varpi+\Delta\varpi}(s,a)}{\pi_{\varpi}(s,a)}$, the objective function is a clipped version instead of KL divergence penalty as,

$$\mathbb{E}_{s\sim\rho^{\pi_{\varpi}},a\sim\pi_{\varpi}}[\min(r_t(\Delta\varpi)A^{\pi_{\varpi}}(s,a), clip(r_t(\Delta\varpi), 1-\epsilon, 1+\epsilon)A^{\pi_{\varpi}}(s,a))], \tag{3.47}$$

where ϵ is a hyperparameter. That function clips the change in probability distribution near rate one, and offers simple implementation with better results [16].

CHAPTER 4

A BI-ROTOR WITH TAIL APPENDAGE

4.1 Dynamics of Tail Assisted Bi-rotor

Tail appendage is a bio-inspired method used in robotics field [18] in order to improve the capabilities of a robotic platform. In Figure 4.1, tail assisted bi-rotor model is represented. At this point, an additional rigid body is connected to the center of bi-rotor body with an actuated revolute joint.

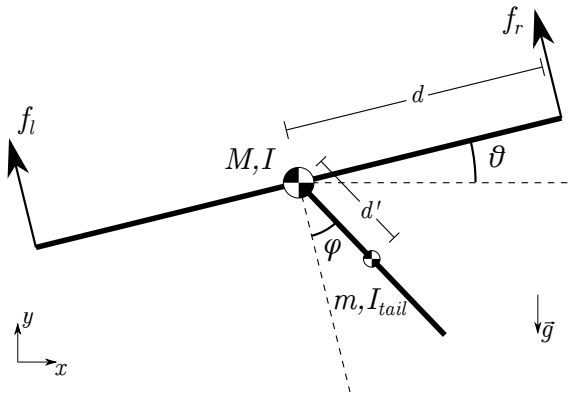

Figure 4.1: Our bi-rotor model with tail appendage

The state vector is now extended with orientation and angular velocity of tail with respect to the bi-rotor body as,

$$\mathbf{x} = [p_x, p_y, \vartheta, v_x, v_y, \omega, \varphi, \dot{\varphi}]^T, \tag{4.1}$$

where φ represents the tail angle with respect to main body as seen in figure. The input vector is extended with torque input of revolute joint as,

$$\mathbf{u} = [f_l, f_r, \tau_{tail}]^T, \tag{4.2}$$

where τ_{tail} represents the torque applied to the revolute joint.

Dynamics of tail assisted bi-rotor system can be derived from Euler-Lagrange formulation. The generalized coordinates of system is defined as;

$$\mathbf{q} = [p_x, p_y, \vartheta, \varphi]^T. \tag{4.3}$$

Position of tail center of mass can be calculated as,

$$p_{x,tail} = p_x + d' \sin(\vartheta + \varphi), \tag{4.4}$$

and,

$$p_{y,tail} = p_y - d' \cos(\vartheta + \varphi), \tag{4.5}$$

where d' is the distance from revolute joint to center of mass of tail. Then, velocities of tail center of mass is found by taking time derivative of equations 4.4 and 4.5 as;

$$v_{x,tail} = v_x + d'(\dot{\vartheta} + \dot{\varphi}) \cos(\vartheta + \varphi), \tag{4.6}$$

and,

$$v_{y,tail} = v_y + d'(\dot{\vartheta} + \dot{\varphi}) \sin(\vartheta + \varphi). \tag{4.7}$$

Kinetic energy K of system is,

$$K = \frac{1}{2}[M(v_x^2 + v_y^2) + I\omega^2 + m(v_{x,tail}^2 + v_{y,tail}^2) + (I_{tail})(\omega + \dot{\varphi})^2], \tag{4.8}$$

where body mass, body inertia, tail mass and tail inertia are M, I, m and I_{tail} respectively. Potential energy U of system is,

$$U = Mgy + mg(y - d' \cos(\vartheta + \varphi)), \tag{4.9}$$

where g is the gravitational acceleration. The Lagrangian is calculated as,

$$L = K - U, \tag{4.10}$$

and the Euler-Lagrange formulation is,

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}} - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{f}, \tag{4.11}$$

where $\mathbf{f}$ is generalized forces vector and defined as;

$$\mathbf{f} = \begin{bmatrix} -(f_r + f_l)\sin(\vartheta) \\ (f_r + f_l)\cos(\vartheta) \\ (f_r - f_l)d \\ \tau_{tail} \end{bmatrix}. \tag{4.12}$$

Rearranging Equation 4.11 and collect double time derivative of generalized coordinates we get,

$$\mathbf{M}\ddot{\mathbf{q}} = \begin{bmatrix} -(f_l + f_r)\sin(\vartheta) + md'(\dot{\vartheta} + \dot{\varphi})^2 \sin(\vartheta + \varphi) \\ (f_l + f_r)\cos(\vartheta) - md'(\dot{\vartheta} + \dot{\varphi})^2 \cos(\vartheta + \varphi) \\ d(f_r - f_l) - mgd' \sin(\vartheta + \varphi) \\ \tau_{tail} - mgd' \sin(\vartheta + \varphi) \end{bmatrix}, \tag{4.13}$$

where $\mathbf{M}$ is the inertia matrix,

$$\mathbf{M} = \begin{bmatrix} M + m & 0 & md' \cos(\psi) & md' \cos(\psi) \\ 0 & M + m & md' \sin(\psi) & md' \sin(\psi) \\ md' \cos(\psi) & md' \sin(\psi) & I + I_{tail} + md'^2 & I_{tail} + md'^2 \\ md' \cos(\psi) & md' \sin(\psi) & I_{tail} + md'^2 & I_{tail} + md'^2 \end{bmatrix}, \tag{4.14}$$

and $\psi = \vartheta + \varphi$.

In Equation 4.13, multiplying both sides by $\mathbf{M}^{-1}$ we get third row as,

$$\ddot{\vartheta} = \frac{d(f_r - f_l) - \tau_{tail}}{I}, \tag{4.15}$$

that gives the angular dynamics of bi-rotor body.

4.1.1 Time-discretization of System Dynamics

System dynamics of bi-rotor is discretized in time in order to simulate. This discretization is held in constant time steps. We show the mathematical description of discretization from a general representation in Equation 3.11. Although dynamics with tail appendage is derived in a different form in Equation 4.13, it can also be represented in a state-space like form. At a discrete time step t we can calculate continuous time derivative of state vector by Equation 3.11 as,

$$\mathbf{ds}_t = A\mathbf{s}_t + B\mathbf{a}_t + \mathbf{g}, \tag{4.16}$$

where $\mathbf{s}_t$ and $\mathbf{a}_t$ are state and action vectors represented in discrete time and $\mathbf{ds}_t$ is vector of time derivatives. Then, for a small time step Δt next state can be calculated as,

$$\mathbf{s}_{t+1} = \mathbf{s}_t + \Delta t \mathbf{ds}_t. \tag{4.17}$$

Note that, although inspired from each other we use different notations for discrete-time states and a state of Markov Decision Process. Former one is a vector and letter one is a tuple containing same values.

This implementation converges to continuous time solution by smaller time-steps. Although there is a small gap between real world scenarios, in the scope of this thesis we apply all of our experiments on this method and compare them in fair.

4.2 Design of Simulation Environment

We design a simulation environment to run the bi-rotor dynamics with and without tail as given in Equations 3.12 and 4.13. Our Reinforcement Learning agent interacts with this environment as shown in the general agent-environment interaction scheme in Figure 3.4. Our simulation environment takes action signal a_t at time step t, runs the bi-rotor dynamics for small time duration Δt, gives the next state s_{t+1} and reward signal r_{t+1}. The transition dynamics from s_t to s_{t+1} under applied action a_t are given in section 3.1 and 4.1. The reward signal is given according to the state distance from goal point as the details will be explained in section 4.2.3.

The simulation environment is implemented at Python programming language. Box2D physics engine [43] is used to simulate dynamics of bi-rotor. This is an open-source engine that solves the differential equations of rigid body dynamics for multiple bodies in two-dimensional space as given in Equation 4.17. We also used OpenAI's gym toolkit [44] to implement a Deep Reinforcement Learning environment software interface. This toolkit is commonly used in Deep Reinforcement Learning field for developing and comparing algorithms.

A gym environment is basically consist of two methods: *step, reset*. *Reset* method initiates the environment according to initially given state distribution which is detailed in Subsection 4.2.1. *Step* method makes one step on environment with given action and returns the next state, the reward signal and termination status. Termination of an episode is detailed in Subsection 4.2.2, and the reward function is given in Subsection 4.2.3. Other relevant parameters implemented in the simulation environment is given in Table 4.2.

In the simulation environment, transition of the bi-rotor during an episode can be rendered optionally. Figure 4.2 shows the render of an instance.

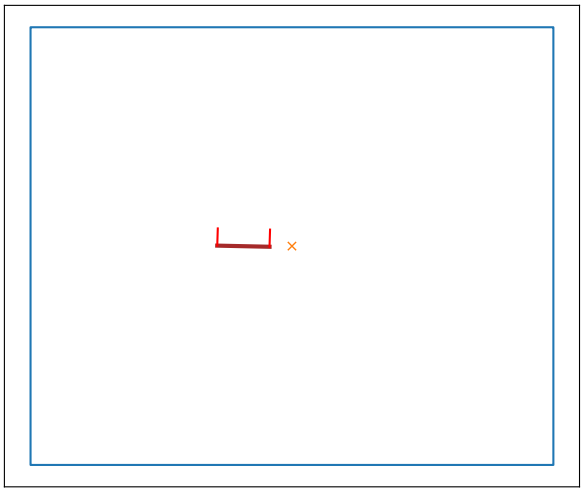

Figure 4.2: An instance of bi-rotor platform flying. Dark red line is body of bi-rotor, light red lines are rotor thrusts, yellow cross is the aiming point. The bi-rotor is restricted in 10x10 square indicated with blue lines. Simulation episode terminated if platform crosses this boundary.

4.2.1 Initial State Constraint

Starting state of each episode is defined by a probability distribution over state space. We use uniform distributions over position to parameterize initial state and limit the Reinforcement Learning problem in a scope. Mathematically, we derive the position terms of state as,

$$|p_x| \sim \mathcal{U}(p_{x,l}, p_{x,h}), \tag{4.18}$$

$$|p_y| \sim \mathcal{U}(p_{y,l}, p_{y,h}), \tag{4.19}$$

where $p_{x,l}, p_{x,h}, p_{y,l}, p_{y,h} \in [0, \infty)$ are parameters representing lower and higher bounds of distribution. Note that, we derive absolute values of this position and they can be negative or positive equally likely.

4.2.2 Termination Criteria

There are two main termination case for episodes. First, if the bi-rotor exits the region so far it terminates. We chose this region as 10x10 square around the goal location, which is shown in Figure 4.2 with blue lines. Mathematically, the criteria for termination is,

$$\left\| \begin{matrix} p_x \\ p_y \end{matrix} \right\|_\infty > 5, \tag{4.20}$$

for this case. This allows the agent to not explore far regions. At the first stages of training, the agent usually drives bi-rotor far regions and we gain time by terminating this irrelevant situations.

Second case is a time limit for episodes. During training after the agent learns to stabilize bi-rotor around goal location, it can drive around there forever. We limit episode time to 4 seconds -400 time steps- for exploration of other states in region.

4.2.3 Reward Shaping

In Reinforcement Learning field mostly rewards are considered sparse. Reward shaping is a concept for propagating this sparse reward to state space by hand. Main purpose of reward shaping is to canalize agent through the goal in high dimensional state spaces. In our work, we use a shaped reward like it was used in literature [12]. Reward at a time step is calculated for the bi-rotor without tail-appendage as,

$$r_t = 25 - (k_p\|p_t\| + k_v\|v_t\| + k_\vartheta\|\vartheta_t\| + k_\omega\|\omega_t\| + k_a\|a_t - a_h\|), \tag{4.21}$$

where r_t denotes the reward at time step t, p_t denotes the position vector $[p_x \;\; p_y]^T$, v_t denotes the velocity vector $[v_x \;\; v_y]^T$, θ_t denotes orientation, ω_t denotes angular velocity, a_t denotes action vector, a_h is hovering actions and k's are constant parameters. So, the agent gets a maximum reward of 25 and gets less by the distance from goal state and action configuration. We use L^1 norm for norm operations. Similarly, reward for the bi-rotor with tail-appendage is calculated as,

$$r_t = 25 - (k_p\|p_t\| + k_v\|v_t\| + k_\theta\|\theta_t\| + k_\omega\|\omega_t\| + k_a\|a_t - a_h\| + k_\varphi\|\varphi_t\| + k_{\dot\varphi}\|\dot\varphi_t\|). \tag{4.22}$$

Parameters of formula is given in Table 4.1. As a comment of parameter selection, main contribution to reward is comes from positional errors. The cost on linear velocities acts as the Derivative term of PID controller, it keeps the agent to oscillate around goal point. It can be noted that also without penalizing angular position or non-sense actions bi-rotor makes angular oscillations around goal point but not changing its position too much.

Table 4.1: Parameters of shaped reward for the bi-rotor with and without tail appendage

Parameters	Bi-rotor	Bi-rotor with tail
k_p	20	20
k_v	2	2
k_θ	1	4
k_ω	0	0
k_a	1	1
k_φ	-	2
$k_{\dot{\varphi}}$	-	0.2

4.2.4 Motor Model

In simulation, instead of supplying commanded thrust forces instantaneously, a simple mathematical motor model is implemented. The thrust forces of propellers according to their motor speeds are defined as,

$$f_l = f_{max}\omega_{m,l}^2, \tag{4.23}$$

and,

$$f_r = f_{max}\omega_{m,r}^2, \tag{4.24}$$

where $\omega_{m,l}, \omega_{m,r} \in [0, 1]$'s are representing normalized motor velocities. Then, the motor velocity commands are derived from normalized actions in Equation 3.26 as,

$$\omega'_{m,l} = \sqrt{\frac{f'_l + 1}{2}}, \tag{4.25}$$

and,

$$\omega'_{m,r} = \sqrt{\frac{f'_r + 1}{2}}, \tag{4.26}$$

where $\omega'_{m,l}, \omega'_{m,r} \in [0, 1]$ are motor commands. Finally, motor velocities are updated through a discrete time low pass filter as,

$$\omega_{m,t} = \frac{4\Delta t}{t_s}(\omega'_{m,t} - \omega_{m,t-1}) + \omega_{m,t-1} \tag{4.27}$$

where $\omega_{m,t-1}$ is motor speed at time $t-1$, $\omega'_{m,t}$ is the motor command, $\omega_{m,t}$ is the motor speed at time t and t_s is the $\%2$ settling time of low pass filter.

Table 4.2: Parameters of the simulation environment

Parameters	Bi-rotor	Bi-rotor with tail
g	9.81	9.81
M	0.8	0.4
m	0	0.4
d	0.5	0.5
d'	-	0.25
Δt	0.01	0.01
$r_{thrust2weight}$	1.5	1.5
t_s	0.15	0.15

CHAPTER 5

CONTROL APPROACH

5.1 Design of a Conventional Controller for Tail Appendage

We have the dynamics of bi-rotor with tail appendage as in Equations 4.13 and 4.14;

$$\mathbf{M}\ddot{\mathbf{q}} = \begin{bmatrix} -(f_l + f_r)\sin(\vartheta) + md'(\dot{\vartheta} + \dot{\varphi})^2 \sin(\vartheta + \varphi) \\ (f_l + f_r)\cos(\vartheta) - md'(\dot{\vartheta} + \dot{\varphi})^2 \cos(\vartheta + \varphi) \\ d(f_r - f_l) - mgd' \sin(\vartheta + \varphi) \\ \tau_{tail} - mgd' \sin(\vartheta + \varphi) \end{bmatrix}, \tag{5.1}$$

and,

$$\mathbf{M} = \begin{bmatrix} M + m & 0 & md' \cos(\psi) & md' \cos(\psi) \\ 0 & M + m & md' \sin(\psi) & md' \sin(\psi) \\ md' \cos(\psi) & md' \sin(\psi) & I + I_{tail} + md'^2 & I_{tail} + md'^2 \\ md' \cos(\psi) & md' \sin(\psi) & I_{tail} + md'^2 & I_{tail} + md'^2 \end{bmatrix}. \tag{5.2}$$

We keep the controller designed for out bi-rotor in Section 3.1 as it is for rotor thrust commands. Although dynamics of the tail is also affecting the dynamics of bi-rotor position by Coriolis and gravitational effects as can be seen from Equation 5.1, those effects can be neglected assuming soft and stable control of the tail. Main advantage of tail is observed in Equation 4.15, rewritten as,

$$\ddot{\vartheta} = \frac{d(f_r - f_l) - \tau_{tail}}{I}. \tag{5.3}$$

It is observed from Equation 5.3 that the torque given to tail can be directly used in attitude control of bi-rotor body as the *Angular PD* block does in Figure 3.3. Without changing the controller structure, another block is added to conventional controller as

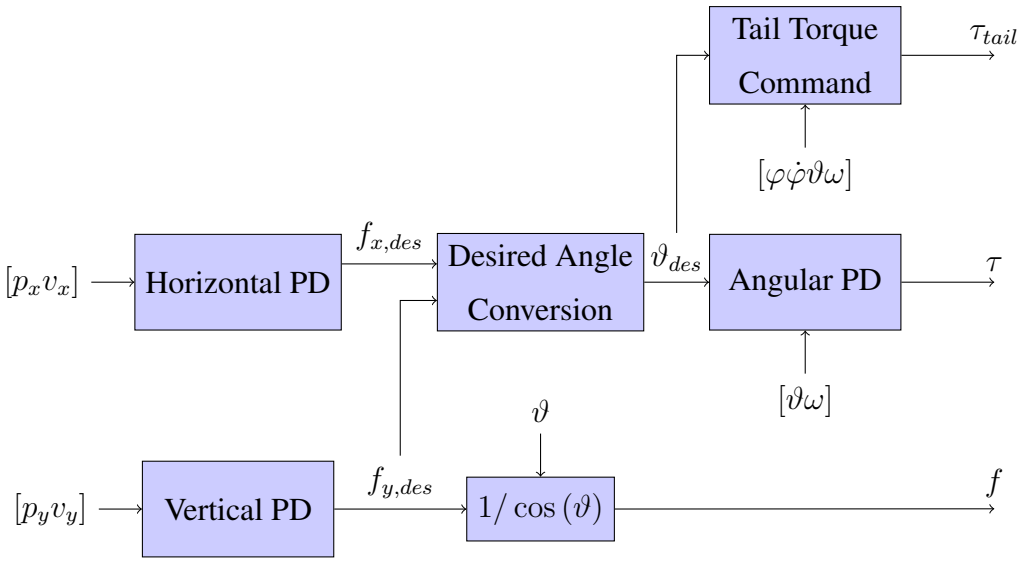

Figure 5.1: Extension of conventional controller for tail appendage

shown in Figure 5.1. In this new block, we can write a PD controller similar to that in Equation 3.20 as,

$$\tau_{tail1} = -(K_{p,\vartheta,tail}(\vartheta_{des} - \vartheta) + K_{d,\vartheta,tail}(-\omega)), \tag{5.4}$$

where τ_{tail1} is a torque command to tail motor and $K_{p,\vartheta,tail}$ and $K_{d,\vartheta,tail}$ are controller parameters. Furthermore, we also want the tail to stay stabilized around its hanging orientation. Another command can be written for that purpose as,

$$\tau_{tail2} = (K_{p,\varphi}(-\vartheta_{des} - \varphi) + K_{d,\varphi}(-\dot{\varphi})), \tag{5.5}$$

where τ_{tail2} is a torque command to tail motor and $K_{p,\varphi}$ and $K_{d,\varphi}$ are controller parameters. We give both commands to the tail motor as,

$$\tau_{tail} = \tau_{tail1} + \tau_{tail2}. \tag{5.6}$$

5.2 Deep Reinforcement Learning Strategies

We train our Deep Reinforcement Learning agent interfacing with the simulation environment using Proximal Policy Optimization [16] method. Main principles of this method was given in section 3.4. We use *stable-baselines* [45] package, an open-source package including implementations of several Deep Reinforcement Learning algorithms, as a base.

5.2.1 Policy Network Structure

A Multilayer Perceptron (MLP) is employed as policy network. Figure 5.2 represents the dimensions of network layers. The input layer is fed with $n_s = 6$ or $n_s = 8$ dimensional state vector according to tail usage. There are two layers of 64 hidden neurons with ReLU activation [46]. Finally, the output layer serves $n_a = 2$ or $n_a = 3$ dimensional action vector with tangent hyperbolic activation that feeds the environment one step. We implement tangent hyperbolic activation at last layer on purpose in order to get normalized actions between negative and positive one.

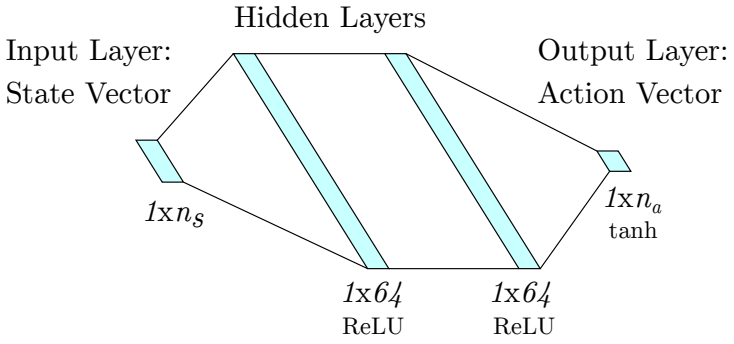

Figure 5.2: Policy network structure. A fully connected neural network: input is state vector, two hidden layers with ReLU activated 64 neurons and output is action vector.

5.2.2 Vanilla Training

We develop different training strategies based on initial state distribution of episodes. We call vanilla training as covering all space initially as it applied in previous work on quad-rotor UAVs [12, 13]. Formally, we assign the constraints in Equations 4.18 and 4.19 as,

$$p_{x,l} = p_{y,l} = 0, \tag{5.7}$$

and,

$$p_{x,h} = p_{y,h} = p_{max} < 5, \tag{5.8}$$

where p_{max} gives the range that the policy finally cover and assigned once from start. The algorithm runs $1e7$ time-steps and saves the best network parameters during training based on total reward collected for a number of previous episodes. Finally, this parameters are used to sample runs of bi-rotor and comparison.

5.2.3 Curriculum Training

As an alternative training strategy we use a curriculum that is increasing the hardness of task gradually. We control p_{max} parameter in Equation 5.8 for adjusting hardness of task during training. This parameter is increases in discrete stages. Each stage is trained for a previously assigned number of time steps and transfers best network parameters to the next stage.

5.2.4 Point-to-point Training

In this case, the task is simplified to fixed horizontal length. By sampling only single starting point initially, the agent becomes more successful when started from that point but does not give guaranty to converge from other states. The constraints are assigned as,

$$p_{y,l} = p_{y,h} = 0, \tag{5.9}$$

and,

$$p_{x,l} = p_{x,h} = p_{p2p} < 5, \tag{5.10}$$

where p_{p2p} is the fixed length that the agent learns to overcome.

CHAPTER 6

EXPERIMENTS & RESULTS

6.1 Optimization of Conventional Control

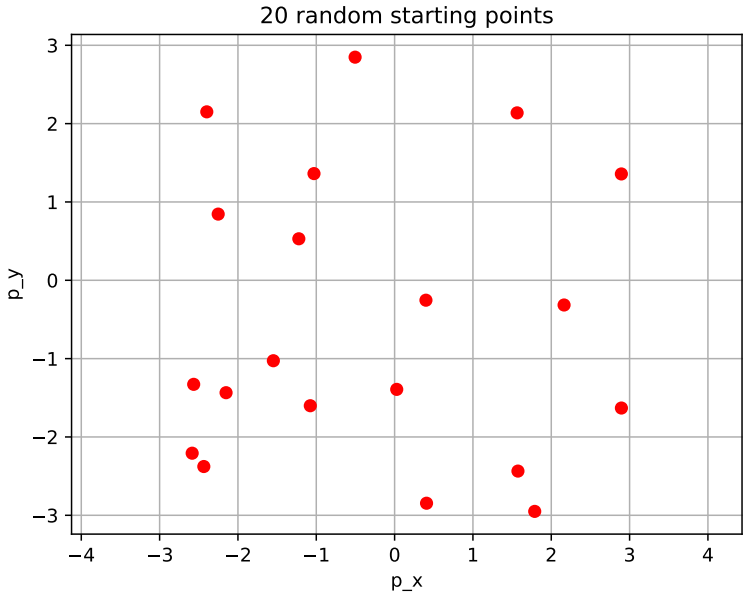

Figure 6.1: Randomly selected points for optimization

Our conventional controller for bi-rotor was presented in 3.1 and the extension for tail equipped case in 5.1. Parameters of controllers are optimized through both global and local optimization techniques. Particle Swarm Optimization (PSO) [47] is employed as global optimizer, and in order to fine tune the local minimum Nelder-Mead [48] method is applied. We use the reward functions in Equations 4.21 and 4.22 to give fair comparison with Deep Reinforcement Learning case. The cost function is defined by the total reward collected through 20 random starting points which are shown

in Figure 6.1. This points are used for all experiments in this section. Resulting parameters of optimization is listed in Table 6.1. Note that, proportional term of tail stabilizer falls negative in optimization. However, the situation does not cause instability due to the derivative term.

Table 6.1: Optimized controller parameters of our conventional controller

Parameter	Bi-rotor	Bi-rotor with tail
$K_{p,x}$	3.23	3.42
$K_{d,x}$	2.57	2.78
$K_{p,y}$	4.45	5.18
$K_{d,y}$	2.89	3.18
$K_{p,\vartheta}$	11.79	4.40
$K_{d,\vartheta}$	2.40	1.11
$K_{p,\vartheta,tail}$	-	3.43
$K_{d,\vartheta,tail}$	-	0.93
$K_{p,\varphi}$	-	-0.80
$K_{d,\varphi}$	-	1.00

6.2 Case 1: Horizontal Step of Bi-rotor

Following, sample runs of bi-rotor with and without tail are presented under conventional and learning based controllers. In each cases, robots are initiated from $(p_x, p_y) = (2, 0)$ point and observed for 4 seconds. For learning based controller, bi-rotors are trained under point-to-point training with episodes starting from $(p_x, p_y) = (2, 0)$ and $(p_x, p_y) = (-2, 0)$. So, they learned to make a 2m horizontal movement from equilibrium point and not guarantied to converge from every close state to center.

6.2.1 Conventional Control of Bi-rotor without Tail

Trajectory followed by the bi-rotor is illustrated in three graphs. First, position and linear velocity of bi-rotor is presented in Figure 6.2. Position in x-direction is settled

in 1.427 seconds considering %2 settling time, and there is a slight change in y-direction is observed due to the attitude change of bi-rotor. By time, all values are converged to zero nicely.

Attitude change of bi-rotor is given in Figure 6.3. Robot first tilts positive side to accelerate in negative x-direction and later the other side to slow down.Finally, force commands are plotted in Figure 6.4. Right and left rotors are derived differently to gain a torque for attitude control. At the end, they equally supply the weight of the bi-rotor.

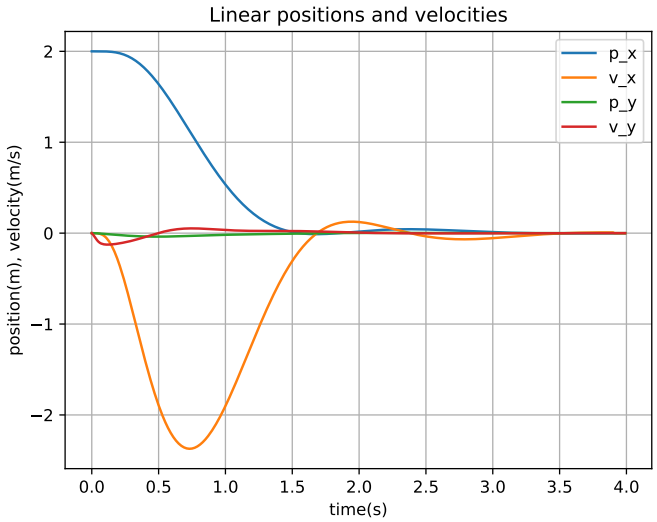

Figure 6.2: Linear position and velocity of bi-rotor in horizontal movement for 4 seconds with conventional controller. Blue, green, orange and red lines represent the position in horizontal and vertical directions and the velocity in horizontal and vertical directions, respectively.

6.2.2 Conventional Control of Bi-rotor with Tail

Trajectory followed by bi-rotor with tail appendage from same starting point of previous case is given in three plots. In Figure 6.5, position and linear velocity is plotted. The settling time of x-position is 1.545 seconds, slightly worse than non-tailed case, and although x-velocity starts decreasing sharper that non-tailed case it finally show similar performance. However, it should be also considered that together with tail,

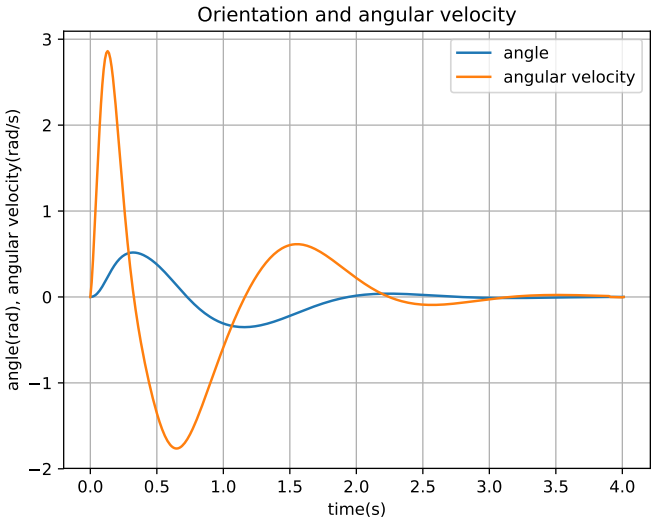

Figure 6.3: Transitions of orientation and angular velocity of bi-rotor in horizontal movement for 4 seconds with conventional controller.

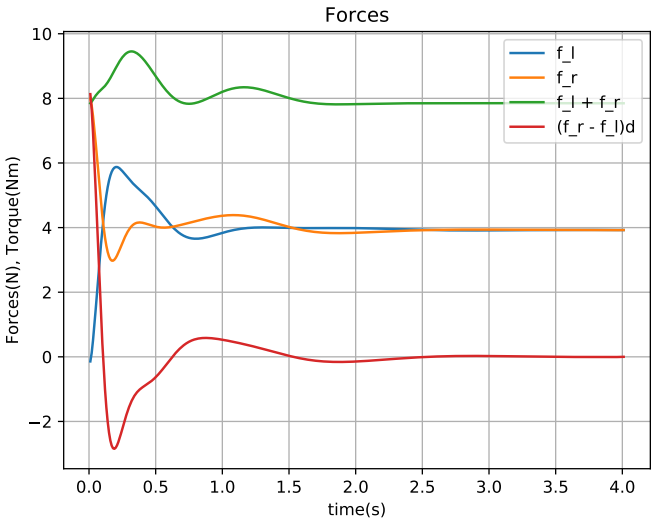

Figure 6.4: Commanded forces, their sum and effective torque of bi-rotor in horizontal movement for 4 seconds with conventional controller. Blue and orange lines represent the commands sent to left and right rotors, respectively. Green and red lines represent equivalent total force and equivalent torque on robot body.

center of mass slides towards it and this makes the robot slower to attitude change. All positions and due to them velocities converged to zero at the end.

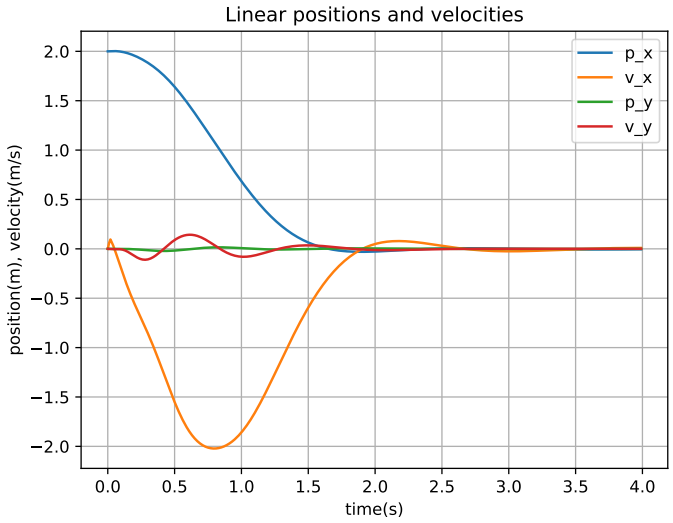

Figure 6.5: Linear position and velocity of bi-rotor with tail appendage in horizontal movement for 4 seconds with conventional controller. Blue, green, orange and red lines represent the position in horizontal and vertical directions and the velocity in horizontal and vertical directions, respectively.

Orientation and angular velocities of body and tail are plotted in Figure 6.6 on the same graph. Body angle and velocity draw a very similar pattern with non-tailed case. Tail makes a negative movement at first to orient forces faster. Later, it is observed to follow the body angle by following its commands. Note that, these behaviours, such as the peaks observed at time 0.5s and 1.5s, helps the robot to orient back in equilibrium by the tail's gravitation.

Finally, the commanded forces are drawn in Figure 6.7. Notably, the torque due to rotors are nearly half way down compared to non-tailed case by the help of tail appendage.

6.2.3 Learning Based Control of Bi-rotor without Tail

Deep Reinforcement Learning agent is trained for twelve million time steps. Mean episode rewards of last 200 episodes collected through training is plotted in Figure 6.8 with its standard deviation. The agent is learned a valuable control policy later than 8 million time steps. Demonstrated policy parameters are chosen by the performance

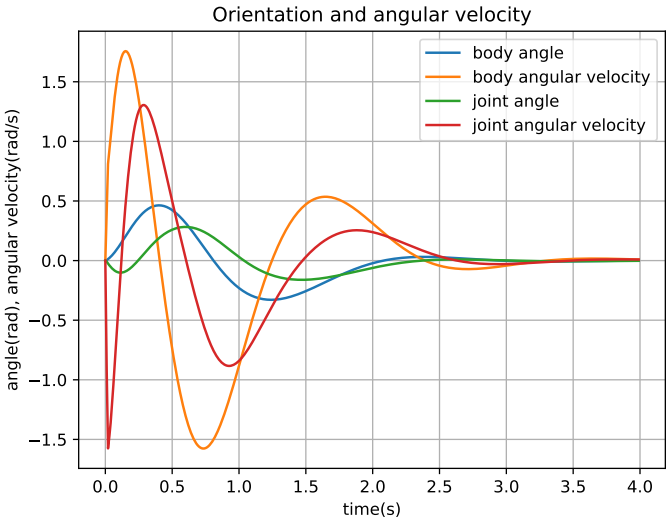

Figure 6.6: Orientation and angular velocity of bi-rotor body and tail in horizontal movement for 4 seconds with conventional controller. Green and orange lines represent orientation and velocity of bi-rotor body. Green and red lines represents orientation and velocity of tail with respect to body.

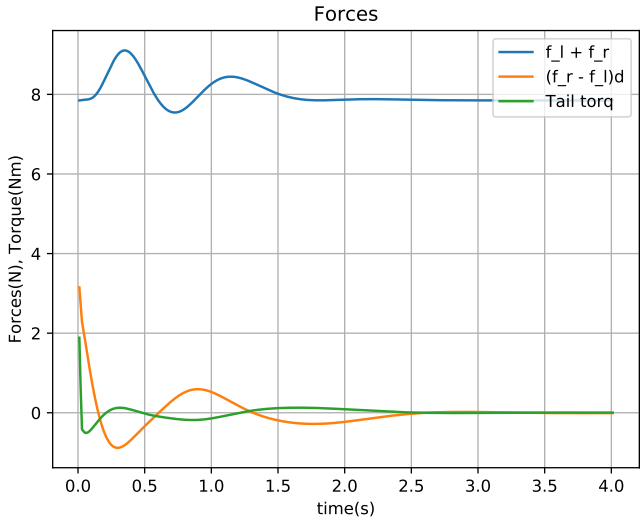

Figure 6.7: Sum of commanded forces, effective torque due to them and tail torque command of bi-rotor with tail appendage in horizontal movement for 4 seconds with conventional controller. Blue and orange lines represent the total force and effective torque due to rotors, respectively. Green line represents torque command to tail.

of last 20 episodes during training.

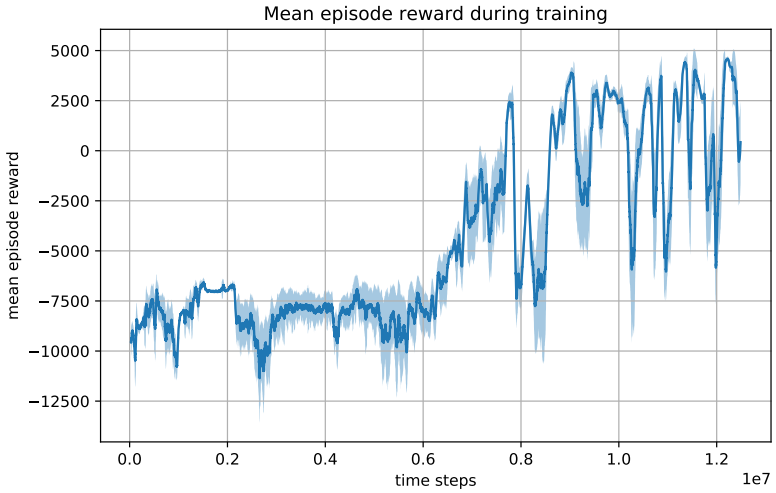

Figure 6.8: Mean episode reward collected during point-to-point training of the bi-rotor without tail. Blue line represents average episode reward in last 200 episodes, and light blue regions are their standard deviation.

Trajectories of bi-rotor without tail under learned control policy is draw in three figures. In Figure 6.9, linear positions and velocities of bi-rotor is presented. It approaches to goal point faster in x-direction when compared to the conventional case. However, it do not stabled in goal point and also a steady state error is left finally. Steady state errors are listed in Table 6.2 under bi-rotor column quantitatively.

Attitude changes are plotted in Figure 6.10. Similar pattern to conventional case is observed for gaining acceleration in x-direction and slow down near goal point. However, changes in angular velocity is more aggressive than conventional case because the neural network can encode a higher order non-linearity and the policy can make distinct changes.

Finally, commanded forces are drawn in Figure 6.11. Notably, changes in commands are more instant than the conventional case as explained before. However, this also means an unnecessary effort looking to total performance in completing the job.

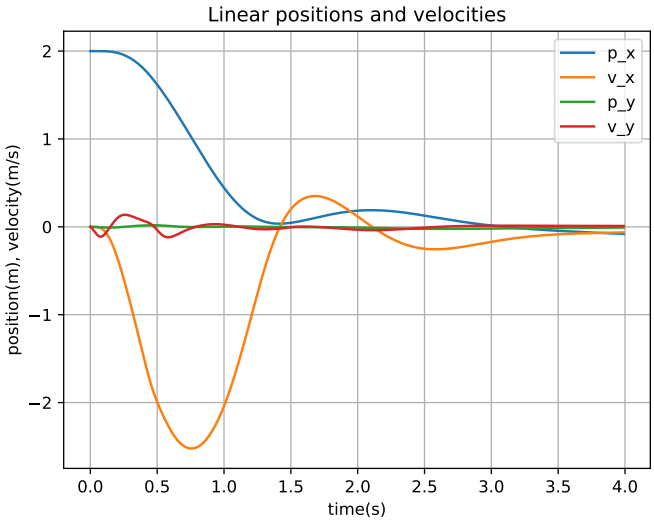

Figure 6.9: Linear position and velocity of bi-rotor in horizontal movement for 4 seconds with point-to-point learned the control policy. Blue, green, orange and red lines represent the position in horizontal and vertical directions and the velocity in horizontal and vertical directions, respectively.

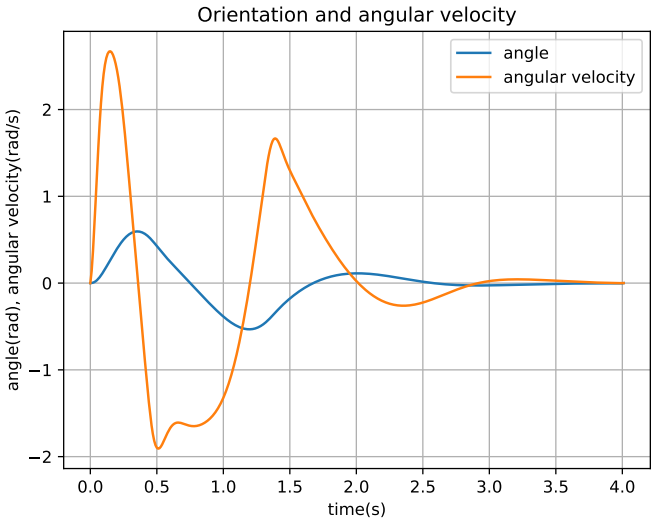

Figure 6.10: Transitions of orientation and angular velocity of bi-rotor in horizontal movement for 4 seconds with point-to-point learned control policy.

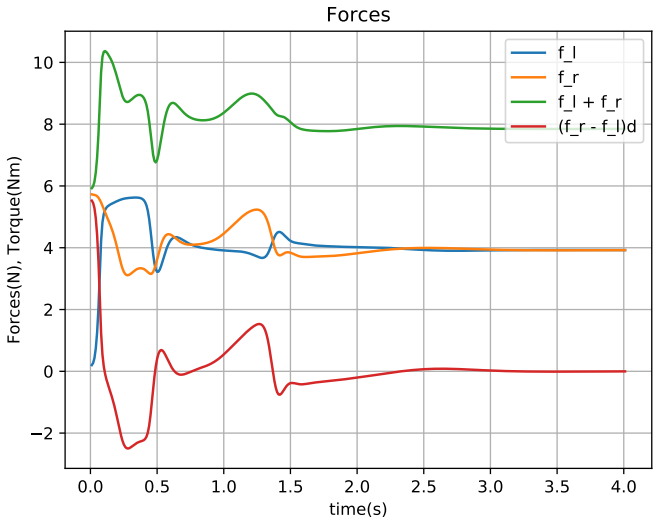

Figure 6.11: Commanded forces, their sum and effective torque of bi-rotor in horizontal movement for 4 seconds with point-to-point learned control policy. Blue and orange lines represent the commands sent to left and right rotors, respectively. Green and red lines represent equivalent total force and equivalent torque on robot body.

6.2.4 Learning Based Control of Bi-rotor with Tail

The bi-rotor with tail is also trained for twelve million time steps. Mean episode rewards of last 200 episodes collected through training is plotted in Figure 6.8 with its standard deviation. The agent is learned a valuable control policy later than 8 million time steps. Demonstrated policy parameters are chosen by the performance of last 20 episodes during training.

Trajectories are again examined in three figures. First, linear position and velocity are plotted in Figure 6.13. It has a rapid initial acceleration in x-direction compared to both non-tailed and tail with the conventional controller case. However, %2 settling time is 1.465s and close to other performances.

Attitude changes of body and tail of bi-rotor are shown in Figure 6.14. Rapid changes is observed like non-tailed control policy because of the same reason. Additionally, the tail is well used during transition. First, it is actuated to rotate body fast and to gain linear acceleration. Later, it is placed such that the gravitational effects help the

Figure 6.12: Mean episode reward collected during point-to-point training of the bi-rotor with tail. Blue line represents average episode reward in last 200 episodes, and light blue regions are their standard deviation.

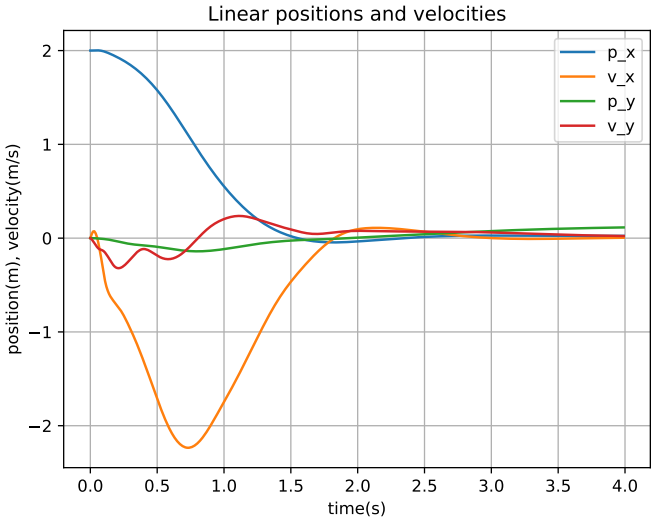

Figure 6.13: Linear position and velocity of bi-rotor with tail appendage in horizontal movement for 4 seconds with point-to-point learned control policy. Blue, green, orange and red lines represent the position in horizontal and vertical directions and the velocity in horizontal and vertical directions, respectively.

bi-rotor to settle in equilibrium. As a bad side, the tail angle is also settles with a steady state error, given in Table 6.2.

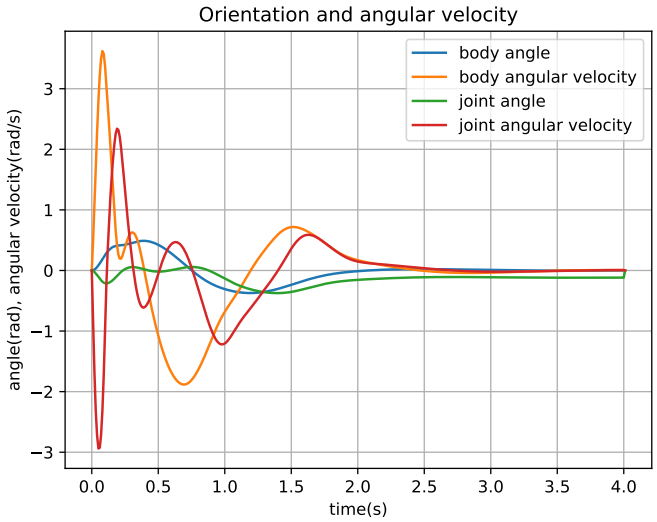

Figure 6.14: Orientation and angular velocity of bi-rotor body and tail in horizontal movement for 4 seconds with point-to-point learned control policy. Green and orange lines represent orientation and velocity of bi-rotor body. Green and red lines represents orientation and velocity of tail with respect to body.

Table 6.2: Steady state errors of point-to-point trained control policies

Steady state error	Bi-rotor	Bi-rotor with tail
p_x	-0.146	0.032
p_y	0.010	0.143
ϑ	0	0
φ	-	-0.113

Finally, force commands are plotted in Figure 6.15. Sudden changes are observed again as a property of neural network policy. It can be also noted that tail helps to rotate body before rotors supply the required torque because of motor delay.

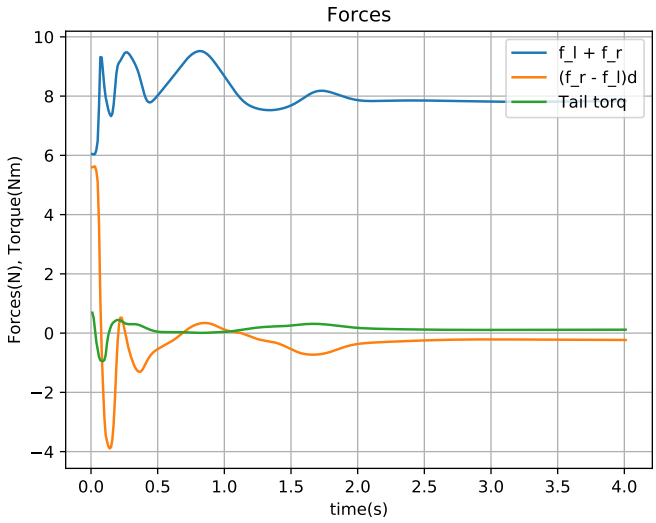

Figure 6.15: Sum of commanded forces, effective torque due to them and tail torque command of bi-rotor with tail appendage in horizontal movement for 4 seconds with point-to-point learned the control policy. Blue and orange lines represent the total force and effective torque due to rotors, respectively. Green line represents torque command to tail.

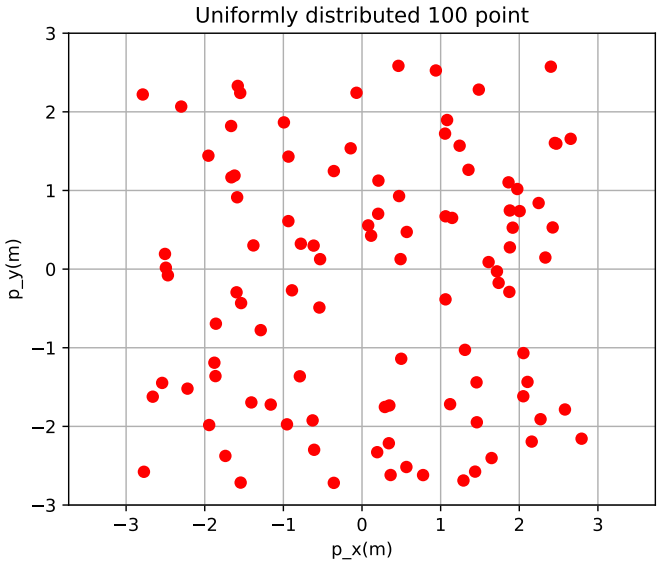

Figure 6.16: 100 random starting points.

6.3 Case 2: Comparison of Controllers through 100 Randomly Started Episodes

In this experiment conventional and learning based controllers are compared on their average performance through 100 episode runs. Here, the learning based controllers are trained with vanilla training with $p_{max} = 3$. Both controllers are started from randomly determined 100 points with and without tail. In Figure 6.16, these starting points are presented. Positional errors are analyzed through those 100 episodes. In order to compare close and far starting points we normalize positional error for each episode between one and zero such that the episode started from one meter and ends at zero.

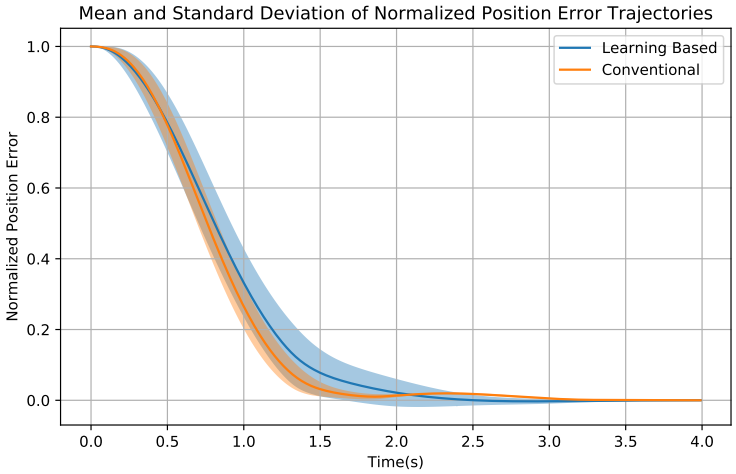

Figure 6.17: Mean and standard deviation of normalized position errors of 100 episodes under conventional and learning based controller without tail.

In Figure 6.17, normalized position errors are plotted for bi-rotor without tail under both controllers. Here, the mean and standard deviation over episodes at each time step is given. This result is an experimental proof of stability of learning based controller, although mathematical analysis is very hard on a neural network policy. However, it can be concluded by comparing standard deviations that, the performance of learning based controller is depending where to start when compared to conventional case.

Similar plot for bi-rotor with tail appendage is given in Figure 6.18. Here, standard deviation of position error for learning based controller is observed even higher than

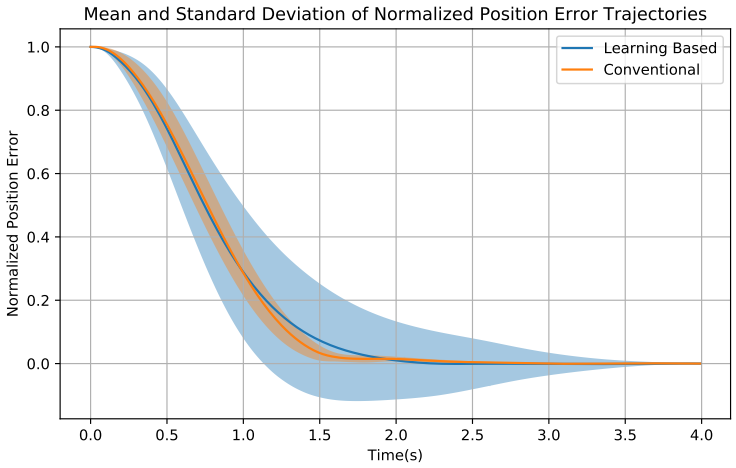

Figure 6.18: Mean and standard deviation of normalized position errors of 100 episodes under conventional and learning based controller with tail.

the case without tail. Settling times of each case is given in Table 6.3 as quantitative results. It can be concluded that the conventional controller is better when comparing settling times.

Table 6.3: Mean settling time of conventional and learning based controllers with and without tail

Controller	%2 Settling Time(s)
Conventional without tail	1.61
Learning Based without tail	2.02
Conventional with tail	1.61
Learning Based with tail	1.86

6.4 Case 3: Curriculum Learning against Vanilla Case

In this experiment we show the advantage of curriculum strategy against vanilla training on bi-rotor without tail. The results of both methods given one by one.

6.4.1 Vanilla

In this setting, the agent is trained with $p_{max} = 3$. Rewards collected in each episode during vanilla training is given in Figure 6.19. In this graph mean of total reward collected in last 200 episode is plotted with its standard deviation. The time-step of best policy chosen by algorithm is represented with orange dot. This policy is chosen with the total performance of last 20 episodes during training.

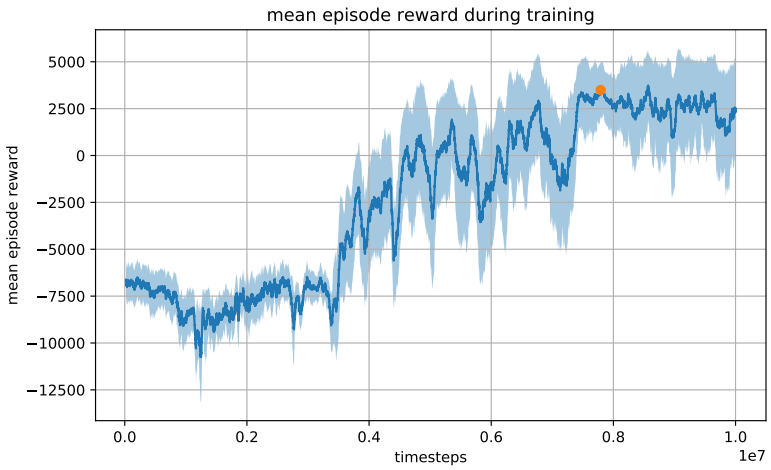

Figure 6.19: Mean reward of an episode collected by agent during vanilla training.

A test run for policy derived by vanilla training is given in Figure 6.20. The bi-rotor is started 2m away in x-axis and its transition observed through positions and velocities in horizontal and vertical directions for five seconds. This starting point is selected because the motion in horizontal is restricted and challenging for bi-rotor platform and vertical motion is straight forward. A significant steady state error can be observed from the figure.

6.4.2 Curriculum

A curriculum learning is conducted with six stages that the parameters are listed in Table 6.4.

Reward collected at each six training stage of curriculum learning is given in Figure 6.21. In this plot different stages represented with different colors, on time scale they

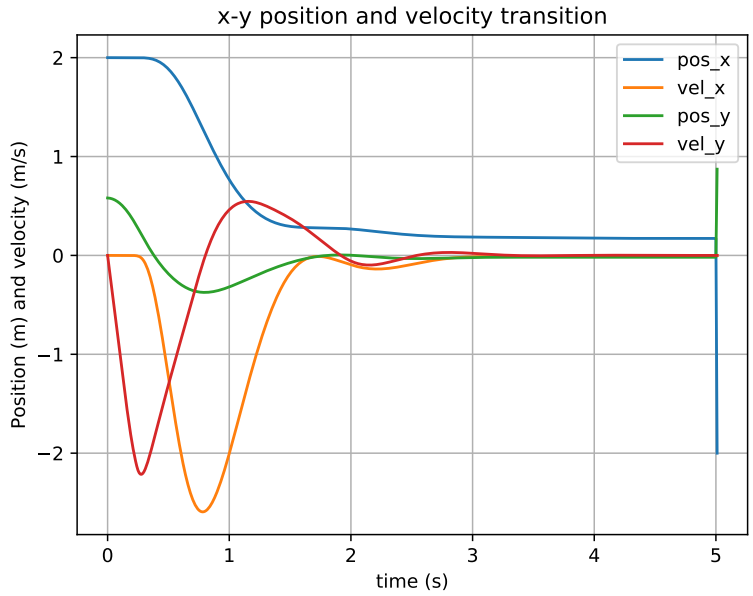

Figure 6.20: Test run for policy learned with vanilla training

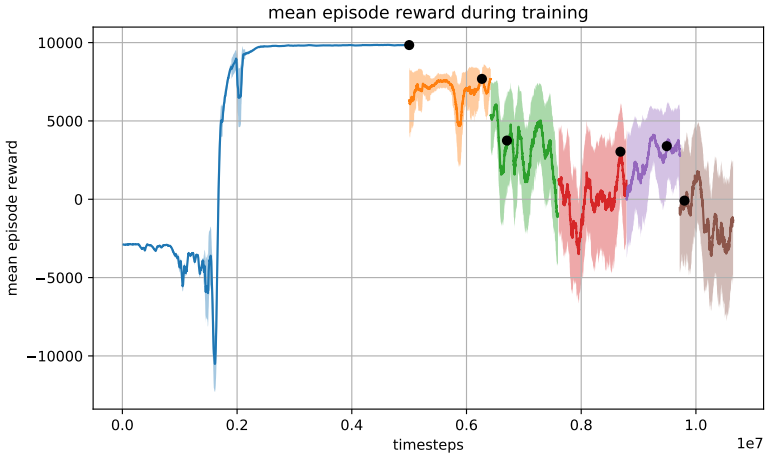

Figure 6.21: Mean reward collected by agent during curriculum training.

are going from 1 to 6. Black dots are representing the time-step that the network is stored and transferred to the next stage. Episode reward at stage 1, it gets 10000 reward after some time-steps, since there is no reward loss caused by state transitions and it learns to stay at the starting point. In other words, it gets very close to 25 reward for 400 time-steps. At higher stages, since the initial starting point of episodes becomes away from goal, the total episode reward can be collected is decreased. Also,

Table 6.4: Stages of Curriculum Learning

Stage	p_{max}	time-steps
1	0	$5e7$
2	1	$1.25e7$
3	1.5	$1.25e7$
4	2	$1.25e7$
5	2.5	$1e7$
6	3	$1e7$

we cannot observe significant improvements at stages 2 to 6 probably because at each stage the policy is generalized to higher stages by neural network. For example, at stage 2 the policy function may learns to cover areas of stage 3 without trained on them.

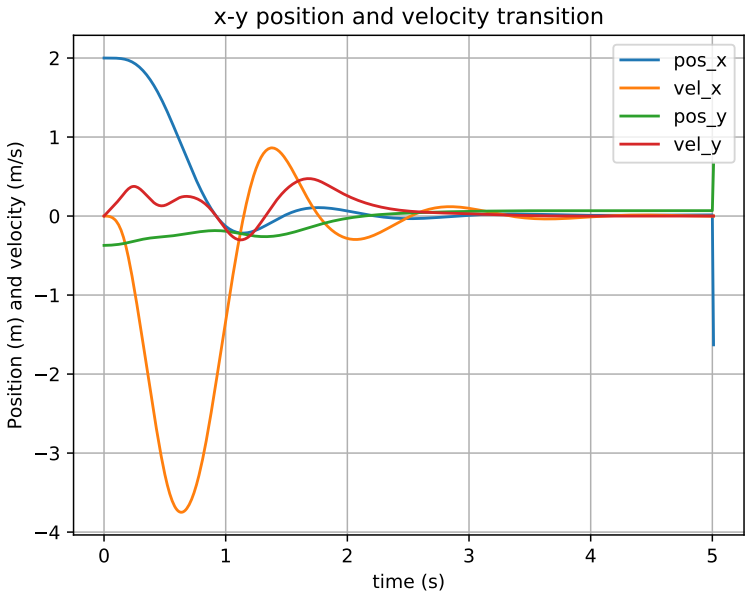

Figure 6.22: Test run for policy learned with curriculum training

In Figure 6.22, a test run for the policy trained with curriculum is given. By comparing with Figure 6.20, this method is more maneuverable -settles in less than 1s- and has much less steady state error. The reason for this improvements may be that the policy first learns to stabilize closer to the goal point and at next stages this behaviour is preserved. However, the policy learned directly from a high state space in vanilla

case.

CHAPTER 7

CONCLUSIONS

In this thesis, control of a bi-rotor, a simplified 2D version of a quad-rotor, was considered. Two kinds of control strategy were studied. First, a control theoretic approach, called conventional method, is employed in a cascaded manner inspired from control methods of quad-rotor. Secondly, a Deep Reinforcement Learning agent is trained to learn a non-linear control policy with trial-and-error. Moreover, the bi-rotor is extended with a torque actuated tail appendage. The bi-rotor with and without tail extension are examined under both controllers.

The conventional controller includes cascaded PD controllers in charge of attitude and positional control. The controller parameters are optimized through a global optimization method, Particle Swarm Optimization, and a local optimization method, Nelder-Mead, for fine tuning. The cost to optimize is rewards defined for learning collected in several episodes to give a fair comparison between learning agent. The learning agent is trained on the simulation environment to stabilize bi-rotor by trial-and-error with only using the reward signals that evaluates how good is the state at the time. A neural network control policy is obtained that maps the current situation of bi-rotor to what action it should apply. In a control theoretic manner, it is a highly non-linear full state feedback mechanism.

The conventional controller requires knowledge on the robot dynamics and a controller design process. On the other side, a simple reward definition can yield a good control policy with Deep Reinforcement Learning. However, the learned policy is not the far best policy instead it was a good local minimum over parameter space of policy neural network. Furthermore, an action of learning policy can differ too much with a small change in input state and also it may not since the neural network

expresses a highly non-linear model. Therefore, learning policy can be more maneuverable than conventional controller. On the other side, the conventional controller is mathematically approved to be stable but the stability analysis of learning policy is complicated. The convergence is also proved in conventional controller but the learning based policy yields a close point to goal in experiments leaving a steady state error behind.

The tail extension provides a fast control chance on attitude. It also provides the opportunity to control center of mass of robot and it can be used in benefit. However, it comes with a complication in robot dynamics. Hence, it is hard to design and implement a conventional controller but learning method does not suffer from that point. It also brings a mechanical complexity for real world applications.

Different learning strategies are also examined in this work by changing the initial state distribution of simulation episodes. It is shown that training under a curriculum from simpler region to harder helps the learning agent to find a better policy. The explanation is that since the parameter space of neural network policy is too high, the curriculum lead it to a better local minimum.

REFERENCES

[1] A. Bitcraze, "Crazyflie 2.0," 2016.

[2] J. Apkarian, "3d helicopter experiment manual," *Canada: Quanser Consulting*, 1998.

[3] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a quad-rotor robot," in *Proceedings Australasian Conference on Robotics and Automation 2006*, Australian Robotics and Automation Association Inc., 2006.

[4] J. Kim, M.-S. Kang, and S. Park, "Accurate modeling and robust hovering control for a quad-rotor vtol aircraft," in *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*, pp. 9–26, Springer, 2009.

[5] A. L. Salih, M. Moghavvemi, H. A. Mohamed, and K. S. Gaeid, "Flight pid controller design for a uav quadrotor," *Scientific research and essays*, vol. 5, no. 23, pp. 3660–3667, 2010.

[6] M. D. Lepine, "Design of a personal aerial vehicle," 2017.

[7] S. Tang and V. Kumar, "Autonomous flying," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 6.1–6.24, 2018.

[8] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 153–158, Ieee, 2007.

[9] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*, pp. 5420–5425, IEEE, 2010.

[10] H. Bou-Ammar, H. Voos, and W. Ertel, "Controller design for quadrotor uavs using reinforcement learning," in *IEEE International Conference on Control Applications (CCA)*, pp. 2130–2135, IEEE, 2010.

[11] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 4653–4660, IEEE, 2016.

[12] 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.

[13] 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.

[14] 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.

[15] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[17] R. Briggs, J. Lee, M. Haberland, and S. Kim, "Tails in biomimetic design: Analysis, simulation, and experiment," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1473–1480, IEEE, 2012.

[18] A. Demir, M. M. Ankaralı, J. P. Dyhr, K. A. Morgansen, T. L. Daniel, and N. J. Cowan, "Inertial redirection of thrust forces for flight stabilization," in *Adaptive Mobile Robotics*, pp. 239–246, World Scientific, 2012.

[19] J. Zhao, T. Zhao, N. Xi, F. J. Cintrón, M. W. Mutka, and L. Xiao, "Controlling aerial maneuvering of a miniature jumping robot using its tail," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3802–3807, IEEE, 2013.

[20] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, ACM, 2009.

[21] 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.

[22] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 2219–2225, IEEE, 2006.

[23] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[24] 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.

[25] 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.

[26] K. Kawaguchi, "Deep learning without poor local minima," in *Advances in neural information processing systems*, pp. 586–594, 2016.

[27] K. Kawaguchi and L. P. Kaelbling, "Elimination of all bad local minima in deep learning," *arXiv preprint arXiv:1901.00279*, 2019.

[28] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, "Automated curriculum learning for neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1311–1320, JMLR. org, 2017.

[29] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," *arXiv preprint arXiv:1707.05300*, 2017.

[30] D. Held, X. Geng, C. Florensa, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," *arXiv preprint arXiv:1705.06366*, 2017.

[31] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher-student curriculum learning," *arXiv preprint arXiv:1707.00183*, 2017.

[32] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5183–5189, IEEE, 2017.

[33] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1765–1772, IEEE, 2013.

[34] S. Daftry, J. A. Bagnell, and M. Hebert, "Learning transferable policies for monocular reactive mav control," in *International Symposium on Experimental Robotics*, pp. 3–11, Springer, 2016.

[35] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[36] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 528–535, IEEE, 2016.

[37] P. J. Huber, "Robust estimation of a location parameter," in *Breakthroughs in statistics*, pp. 492–518, Springer, 1992.

[38] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[39] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*, vol. 2050. Princeton university press, 2015.

[40] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[41] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[42] S. M. Kakade, "A natural policy gradient," in *Advances in neural information processing systems*, pp. 1531–1538, 2002.

[43] E. Catto, "Box2d: A 2d physics engine for games," 2011.

[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[45] A. Hill, A. Raffin, M. Ernestus, A. Gleave, 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.

[46] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[47] 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.

[48] 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.