University of São Paulo

São Carlos School of Engineering

Department of Aeronautical Engineering

Author: **Caio Augusto Zagria Barbosa**

Title: Reinforced Learning for UAV Attitude Control

São Carlos

2019

Author: Caio Augusto Zagria Barbosa

Title: Reinforced Learning for UAV Attitude Control

Monograph presented to the Aeronautical Engineering Course of the São Carlos School of Engineering of the University of São Paulo, as part of the requirements to obtain the title of Aeronautical Engineer.

Teacher Advisor: Prof. Dr. Jorge Henrique Bidinotto

Versão Corrigida

São Carlos

2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Z111r

Zagria Barbosa, Caio Augusto
Reinforced Learning for UAV Attitude Control / Caio Augusto Zagria Barbosa; orientador Jorge Henrique Bidinotto. São Carlos, 2019.

Monografia (Graduação em Engenharia Aeronáutica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2019.

1. UAV. 2. Control Techniques. 3. Reinforced Learning. 4. PID. 5. Técnicas de Controle. 6. Aprendizagem de Máquina. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

ERRATUM

Erratum			
Page	**Line**	**Where is written**	**Should be**
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –
– –	– –	– –	– –


Empty Page

ASSESSMENT OR APPROVAL SHEET

Candidato: Caio Augusto Zagria Barbosa
Título do TCC: Reinforced Learning Applied for UAV Attitude Control
Data de defesa: 21/11/2019

Comissão Julgadora	**Resultado**
Professor Titular Eduardo Morgado Belo	Aprovado
Instituição: EESC - SAA	
Professor Titular Glauco Augusto de Paula Caurin	Aprovado
Instituição: EESC - SAA	

Presidente da Banca: Professor Titular Eduardo Morgado Belo

(assinatura)

RESUMO

Barbosa, C. A. Z. **Reinforced Learning for UAV Attitude Control**. 2019. 81 f. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019

Transportar pessoas é uma tarefa muito complexa e que exige segurança. Um dos principais problemas é que essa tarefa exige um controlador de atitude extremamente confiável. Entretanto, projetar um controlador de atitude para quadricópteros de alto desempenho e confiabilidade não é uma tarefa trivial, pois seu modelo físico possui um alto grau de complexidade: é desejável que os controladores de vôo para drones de passageiros possam tolerar falhas; adaptar-se às mudanças na carga útil e / ou no ambiente; e otimizar a trajetória de vôo. O desenvolvimento de sistemas inteligentes de controle de vôo é uma área ativa de pesquisa, especificamente através do uso de redes neurais artificiais, uma opção atraente, pois são aproximadores universais e resistentes ao ruído. Através de simulações usando o ambiente de desenvolvimento em Python, estudamos a exatidão e precisão do controle inteligente de atitude treinados usando DDPG (Deep Deterministic Policy Gradients) com *Reinforced Learning* por Ator-Crítico. O sistema de controle RL será dinâmico, alterando as constantes do PID para tentar estabilizar o quadricópter. O ator e o crítico serão 2 redes neurais densamente conectadas e distintas, com 2 camadas ocultas. Os resultados serão comparados com o simples controle PID ajustado usando o método de busca na grade de parâmetros para selecionar os melhores ganhos. Embora nos concentramos especificamente na criação de controladores para um quadricóptero, os métodos desenvolvidos por este trabalho aplicam-se a uma ampla gama de aeronaves não tripuladas com vários rotores e também podem ser estendidos a aeronaves de asa fixa. No final, este trabalho nos mostrou que é possível controlar um quadricóptero usando técnicas de *Reinforced Learning*. No entanto, o modelo convergiu apenas para ângulos pequenos, o que torna impossível comparar o resultado do PID original com os resultados do controlador RL. No entanto, não podemos dizer que o PID é melhor que o RL ou o contrário é verdadeiro porque a estabilidade do RL e a resposta a condições não lineares, alterações na carga útil e perturbações externas não foram testadas. Para trabalhos futuros, além de testar a estabilidade do controlador RL, é recomendável tentar a pesquisa numa grade de parâmetros para otimizar o hiperparâmetros , bem como a normalização de minilote.

Palavras-chave: UAV. Técnicas de Controle. Reinforced Learning. PID.

Empty Page

ABSTRACT

Barbosa, C. A. Z. **Reinforced Learning for UAV Attitude Control**. 2019. 81 f. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

Carrying people is a very complex and safety demanding task. One of the main problems is that this task demands a good reliable attitude controller. Therefore, designing an quadcopter attitude controller with superior performance, is not a trivial task as its physical model has some high degree of complexity: is desirable that passenger UAV flight controllers are able to tolerate faults; adapt to changes in the payload and/or the environment; and to optimize flight trajectory, to name a few. The development of intelligent flight control systems is an active area of research, specifically through the use of artificial neural networks which are an attractive option given they are universal approximators and resistant to noise. Through simulations using Python environment, we study the accuracy and precision of attitude control provided by intelligent flight controllers trained using Deep Deterministic Policy Gradients (DDPG) with Actor-Critic Reinforced Learning. The RL control system will be dynamic changing the PID constants while trying to stabilize the quadcopter. The actor and critic will be 2 different fully dense connected neural networks with 2 hidden layers. Results will be compared over simple PID control tuned using parameter grid search method to selecting the gains. While we specifically focus on the creation of controllers for a quadcopter, the methods developed hereby apply to a wide range of multi-rotor UAVs, and can also be extended to fixed-wing aircraft. At the end, this work has shown us that is possible to control a quadcopter using Reinforced Learning techniques. However, the model only converged for small angles, what makes impossible to compare the original PID result with the Reinforced Learning results. However, we cannot say that PID is better than RL or the opposite is true because the RL stability and response to non-linear conditions, payload change as well as external perturbations were not tested. For future works, in addition to testing the RL stability, is recommended to try parameter grid search optimization for the Reinforced Learning control as well as mini-batch normalization.

Keywords: UAV. Control Techniques. Reinforced Learning. PID.

INDEX OF FIGURES

INDEX OF TABLES

TABLE OF CONTENTS

1. Introduction

Before proceed with a brief history of UAVs and quadcopters, let's see 2 important definitions. These definitions were found and copied online from 'Drone and Quadcopter' website: [1]

- **"Drone"** is a broad term used to describe any kind of unmanned aerial vehicle (UAV). As such, it can be used to describe both UAVs that are remotely controlled and those that are controlled by onboard computers. These types of aerial craft can look either like a small airplane or like a helicopter. They generally have two characteristics that set them apart as drones: They are engine-controlled, and they can fly for long periods of time.

- **"Quadcopter"** is a more specific term used to refer to a drone that is controlled by four rotors. It is also called a quadrotor or a quadrotor helicopter. The rotors on the quadcopter each consist of a motor and a propeller. In addition, these UAVs are always controlled remotely instead of being controlled by a pre-programmed, onboard computer. Quadcopters resemble helicopters, but balance themselves by the movement of the blades and not by the use of a tail rotor. It is also a subcategory of "multirotor".

Figure 1.1: General Atomics MQ-9 Reaper – Fixed Wing (subcategory) UAV

Source: Wikipedia (2019)

Figure 1.2: Quadcopter (Q450 Model based) – Subcategory of UAV

Source: Elecbits (2019)

The first stop in our drone history timeline is the very early history of drones. By this definition, the earliest unmanned aerial vehicle in the history of drones was seen in 1839, when Austrian soldiers attacked the city of Venice with unmanned balloons filled with explosives. [2]

Some of these Austrian Balloons were successful, but a number of them blew back and bombed the Austrians' own lines, so the practice did not become widely adopted (clever decision). However, the invention of winged aircraft changed everything for manned and unmanned vehicles alike.

Figure 1.3: Bombs over Venice - First UAVs in history

Source: History Today (Volume 8 Issue 6 June 1958)

Already in 1951, in the USA, the first modern UAV, known as Firebee (Q2A), appears. Using a rudimentary datalink, it was remotely controlled by the operator aboard a nearby military aircraft. And their mission was to help train fighter pilots, helping them adapt to the new generation of modern aircraft and weaponry for intercepting enemy aircraft. Also, it is observed that such UAVs did not have any control system for their self-stabilization, depending only on the expertise of their operator.

Figure 1.4: The Q2A/C targets were the first unmanned drone targets used by the Navy at WSMR starting in 1959.

Source: White Sands Missile Range Museum

So, following the escalating tensions of the end of the Cold War and the Vietnam War, the US attempted to develop new UAVs for aerial surveillance and attack, but was unsuccessful. They therefore upgraded the Firebee UAVs, increasing their autonomy to up to 8 flight hours and receiving, for the first time, a control system that allowed pre-programmed missions and even autonomous flight on some routes. These were called the BQM-34 Firebee and, launched from the C-130 Hercules, were widely used successfully during the Vietnam War and the Yom Kippur War for aerial surveillance and as baits for the discovery of anti-aircraft batteries. Its last use occurred in the operation "Iraq Freedom" which began in 2003 after the 9/11 attacks.

Figure 1.5: BQM-34 Firebee ready to be lunched from a C-130 Hercules

Source: Wikipedia (2019)

In 1995, the Predator RQ-1L UAV (General Atomics) was the first deployed UAV to the Balkans and was proved very effective for surveillance and tracking targets.

The terrorist attacks on the US in 2001 led to the so-called 'war on terror' and a decisive shift in the military strategy of the US and its allies. The war on terror has been a battle waged against asymmetric opposition – usually small groups, or even individuals, who may be dispersed, highly mobile and located in remote locations. The US response to these challenges has been a policy of persistent surveillance and a significant increase in the speed and versatility of attacks: developing capabilities for persistent surveillance, tracking, and rapid engagement.

Figure 1.6: Predator RQ-1L UAV

Source: General Atomics

Armed Predator operational flights over Afghanistan began on 7 October 2001 [7] with the first Predator drone strike taking place in early November 2001. Details of this first strike, like much that information about drone wars, is swathed in secrecy and confusion. In the first two months of operations in Afghanistan, some 525 targets were laser designated by Predators and, according to Pete Singer, author of Wired for War, ""the generals who once had no time for such systems couldn't get enough of them."[8]

A year later in November 2002 the first lethal operation using a Predator drone took place in Yemen. This time there were no other aircraft involved, just a Predator being controlled by a pilot sitting at Camp Lemmonier in Djibouti. [9]

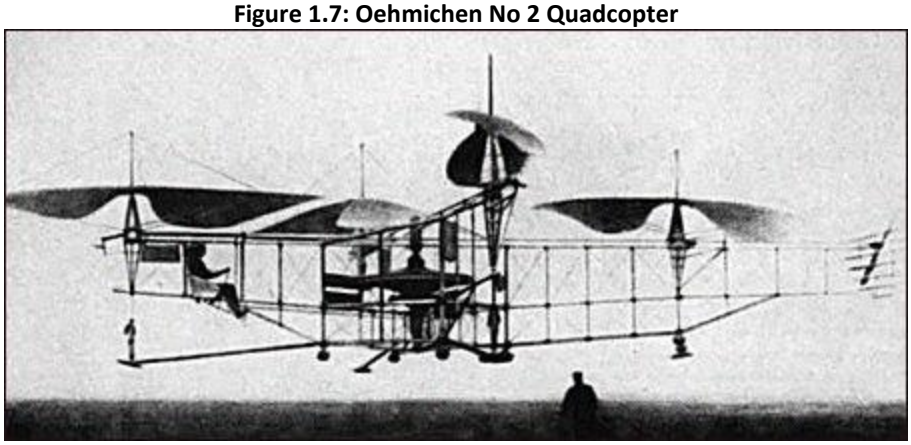
Greatly accelerated by the use of UAVs in the American war in Afghanistan (2001), there was given rise to a number of researches that eventually created the new and modern UAVs and their diverse areas of activity, not just military, as we know it today [4].

According to a recent report by Goldman Sachs [11], military spending will remain the main driver of drone spending in the coming years. Goldman estimates that global militaries will spend $70 billion on drones by 2020, and these drones will play a vital role in the resolution of future conflicts and in the replacement of the human pilot.

But, what about the quadcopters in this context? Quadcopters were among the first vertical take-off and landing vehicles (VTOLs) and its history goes back to 1920 with Oehmichen 2, invented by Etienne Oehmichen. This aircraft made 1000 successful flights and flew a recorded distance of 360 meters.

Earlier helicopters used tail rotors to counterbalance the torque, or rotating force, generated by a single, main rotor. The tail rotor on a single rotor helicopter design consumes between 10 and 15% of the engine power, yet it creates no lift or forward thrust. Part of the main rotor rotates over the fuselage, pushing down washed air against it, reducing effective lift. This was wasteful and inefficient. Engineers developed quadcopters to solve the problems that helicopter pilots had with making vertical flights.

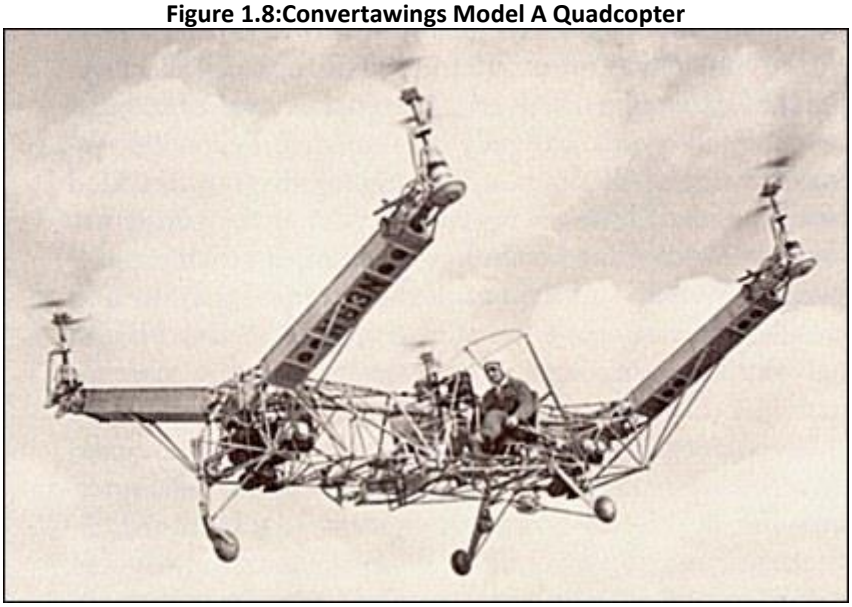
Figure 1.7: Oehmichen No 2 Quadcopter



Source: Wikipedia (2019)

Around the same time George de Bothezat built and tested his quadcopter for the US army, completing a number of test flights before the program was scrapped. The 'Convertawings Model A' quadcopter designed by Dr. George E Bothezat, appeared in 1956. It was the first to use propulsion, or a propeller's forward thrust, to control an aircraft's roll, pitch and yaw. The Curtis Wright V27, developed by the Curtis Wright Company, followed in 1958.

Figure 1.8:Convertawings Model A Quadcopter



Source: Aviastar (1969)

Early quadcopters would typically have the engine sitting somewhere centrally in the fuselage of the copter, driving the 4 rotors via belts or shafts. Belts and shafts however are heavy and importantly, subject to breakage. As the 4 rotors of a quadcopter are all slightly different from each other, a quadcopter is not naturally stable, simply running 4 rotors at the same speed, while producing enough

lift to hover the copter, does NOT produce stable flight. On the contrary, quadcopters have to be constantly stabilized. In the absence of computers, this meant a monumental workload for the pilot. [16]

As a result, multicopter designs were abandoned in favor of single, or on rare occasions for very large transport helicopters, double rotor designs.

However, with the advent of electric motors and especially microelectronics and micromechanical devices, a few years ago, it became possible to build reliable and efficient multirotor. Modern multicopters have an electric motor mated to each rotor, sitting directly below or above it. A flight computer constantly monitors the orientation of the copter and corrects for instability by changing not the pitch of the rotors but simply the rpm of the individual motors/rotors. This fixed pitch design is much simpler than the complex swashplate mechanics that are required for single rotor helicopters.

Today you can buy quadrotor drones—also known as quadcopters—of just about any kind, for just about any price. The extremely wealthy can buy gold-plated quads, and the rest of us can buy tiny plastic ones. And, most important for this work, the scaling up of this to aircraft that are able to carry people has only just begun. [15]

Figure 1.9: First Electric Passenger Drone - Ehang 184

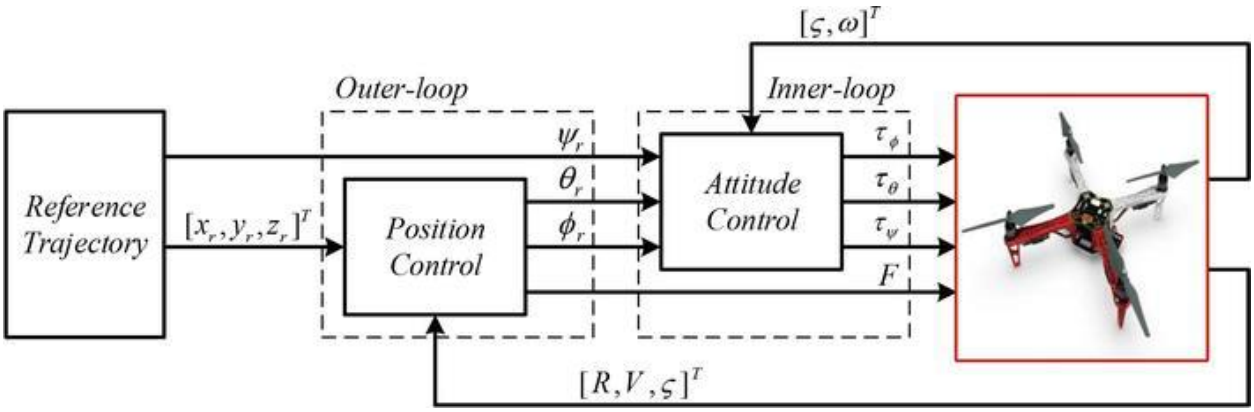
Source: Wikipedia (2019)

Carrying people is a very complex and safety demanding task. There are many rules and regulatory agencies analyzing this problem before finally allow people transportation using quadcopters or multirotor, and thus today there is not certified multirotor for this task. One of the main problems is that this task demands a good reliable attitude controller. Therefore, designing an attitude controller with superior performance is one of the most common and important efforts that researchers all around the world are nowadays undertaking. [17]

Controlling the quadcopter dynamics is not a trivial task as its physical model has some high degree of complexity. Santos et al. [18] presented an algorithm to accomplish quadcopter trajectory tracking tasks by controlling the altitude through an adaptive dynamic controller that was capable of dealing with uncertainties in model parameters. Jayakrishnan [19] and Xiong et al. [20] used the sliding mode control (SMC) technique to control the horizontal position and attitude while also providing a significant improvement of altitude control. In other studies [21–23], the second order SMC method was also used to improve quadcopter altitude control performances. Yet another method was presented by Muliadi et al. [24], where the authors proposed a neural network approach to control UAV altitude dynamics. The results obtained with this method were verified through comparisons with a conventional proportional–integral–derivative (PID) control system. However, these approaches [18–24] have a common disadvantage in that the SMC technique generates a high chattering control signal method which reduces the lifetime of the entire system.

Although several control methods have been proposed in the literature, PID control has become the most widely used technique in a variety of applications all over the world because it is simple and easy to design and typically delivers a satisfactory performance. The PID approach was used in many studies [25–31] to achieve not only quadcopter altitude control, but also attitude stabilization and horizontal position control. In recent studies [32–35], the authors proposed the use of a multi-loop control architecture (i.e., inner-loop and outer-loop) to control quadcopters in specific applications. The outer-loop controllers were designed in different ways while the inner-loop controllers were all implemented using the PID control law.

Figure 1.10: Inner-loop and Outer-loop applied for UAV control (illustrative)



Source: Joukhadar, Abdulkader et al. (2019)

Nevertheless, the conventional PID controller has several limitations. First, their fixed gains limit system performance over a wider operational range. When the required range of operation is large, the conventional PID controller is prone instability, because the nonlinearities in the system cannot be properly dealt with. Second, as conventional PIDs are based on a linear model, their performance may suffer in a nonlinear system like a quadcopter. Several studies have attempted to overcome these shortcomings. Phi et al. [36] presented a gain scheduling PID controller which determines the PID gains

by linearly adjusting the gain as a function of tracking errors. In another approach [37], the authors used a pickup table to schedule the PID gains in a quadrotor fault tolerant control task. Both methods were able to improve the control performance under different operating conditions. However, it is still a linear control law which means that it may not perform well in non-linear systems. Furthermore, scheduling the gains results in discontinuous transitions which may result in sudden jerks or oscillations.

The references indicate that most of the existing methods either are complex to design and implement or require great computational resources. Meanwhile, PID control law appears to play an important role for finding a simple and efficient control method for a variety of systems. When exposed to unknown dynamics (e.g. wind, variable payloads, voltage sag, etc), a PID controller can be far from optimal and unsafe for people transportation [45].

However is desirable that passenger UAV flight controllers are able to tolerate faults; adapt to changes in the payload and/or the environment; and to optimize flight trajectory, to name a few. So, a simple PID controller would not achieve the necessary performance. What about a intelligent flight controller?

The development of intelligent flight control systems is an active area of research [46], specifically through the use of artificial neural networks which are an attractive option given they are universal approximators and resistant to noise [47].

Online learning methods (e.g. [48]) have the advantage of learning the aircraft dynamics in real-time. The main limitation with online learning is that the flight control system is only knowledgeable of its past experiences. It follows that its performances are limited when exposed to a new event. Training models offline using supervised learning is problematic as data is expensive to obtain and derived from inaccurate representations of the underlying aircraft dynamics (e.g. flight data from a similar aircraft using PID control) which can lead to suboptimal control policies [49], [50], [51].

An alternative to supervised learning for creating offline models is known as reinforcement learning (RL). In RL an agent is given a reward for every action it makes in an environment with the objective to maximize the rewards over time. Using RL could make possible to develop optimal control policies for a UAV without making any assumptions about the aircraft dynamics.

In this context, through simulations using Python environment, will be done the study of accuracy and precision of attitude control provided by intelligent flight controllers trained using Deep Deterministic Policy Gradients (DDPG) with Actor-Critic Reinforced Learning. The RL control system will be dynamic changing the PID constants while trying to stabilize the quadcopter. The actor and critic will be 2 different fully dense connected neural networks (MLP) with 2 hidden layers. Results will be compared over simple PID control tuned using parameter grid search method to selecting the gains. While this work specifically focus on the creation of controllers for a quadcopter, the methods developed hereby apply to a wide range of multi-rotor UAVs, and can also be extended to fixed-wing aircraft.

2. Background

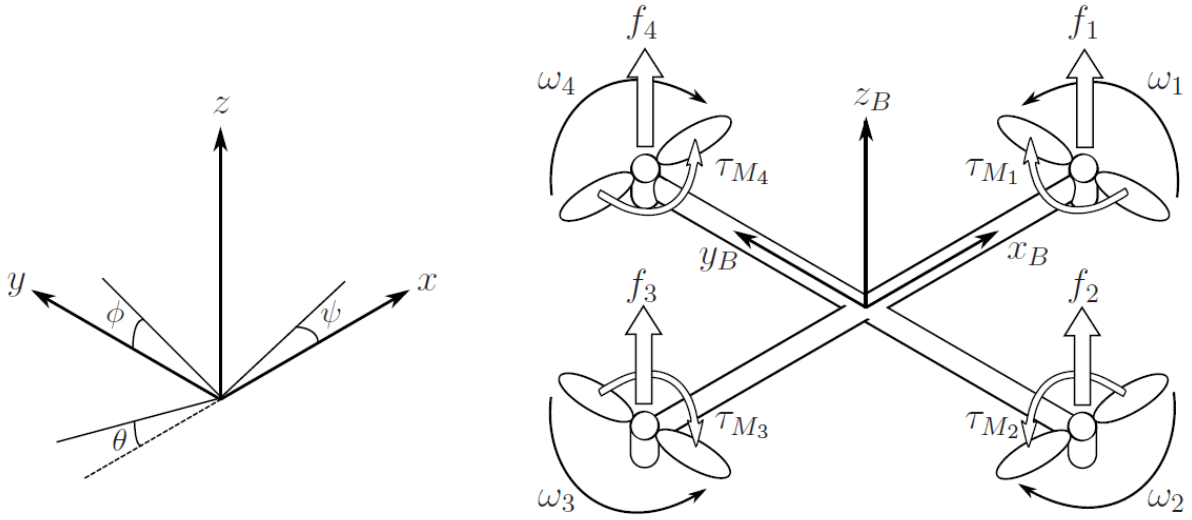
In this chapter, some background knowledge will be explained. Even if you already know the concepts shown here, is always good to remember ☺.

2.1 Quadcopter Mathematical Modelling

This item presents the differential equations of the quadcopter dynamics. They are derived from the Newton-Euler equations. Luukkonen [52] made a very clear and nice paper about modelling the quadcopter dynamics, his job will be reproduced here.

The quadcopter structure is presented in Figure 2.1 including the corresponding angular velocities, torques and forces created by the four rotors (numbered from 1 to 4).

Figure 2.1: The inertial and body frames of quadcopter



Source: Luukkonen (2011)

The absolute linear position of the quadcopter is defined in the inertial frame x, y, z – axes with $\boldsymbol{\xi}$. The attitude, i.e. the angular position, is defined in the inertial frame with three Euler angles $\boldsymbol{\eta}$. Pitch angle $\boldsymbol{\theta}$ determines the rotation of the quadcopter around the y-axis. Roll angle $\boldsymbol{\Phi}$ determines the rotation around the x-axis and yaw angle $\boldsymbol{\psi}$ around the z-axis. Vector $\boldsymbol{q}$ contains the linear and angular position vectors

Equation 1

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad \eta = \begin{bmatrix} \Phi \\ \theta \\ \psi \end{bmatrix}, \qquad q = \begin{bmatrix} \xi \\ \eta \end{bmatrix}.$$

The origin of the body frame is in the center of mass of the quadcopter. In the body frame, the linear velocities are determined by $\boldsymbol{V_B}$ and the angular velocities by $\mathbf{v}$

Equation 2

$$V_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}, \qquad \mathrm{v} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}.$$

The rotation matrix from the body frame to the inertial frame is:

Equation 3: Rotation matrix

$$\boldsymbol{R} = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix}$$

In which $\boldsymbol{S_x} = \mathbf{sin}(\boldsymbol{x})$ and $\boldsymbol{C_x} = \mathbf{cos}(\boldsymbol{x})$. The rotation matrix $\boldsymbol{R}$ is the orthogonal thus $\boldsymbol{R^{-1}} = \boldsymbol{R^T}$ which is the rotation matrix from the inertial frame to the body frame.

The transformation matrix for angular velocities from the inertial frame to the body frame is $\boldsymbol{W_\eta}$, and from the body frame to the inertial frame is $\boldsymbol{W_\eta^{-1}}$, as shown in [53],

Equation 4

$$\dot{\eta} = W_\eta^{-1}\mathrm{v} \ , \qquad \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi / C_\theta & C_\phi / C_\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$$\mathrm{v} = W_\eta \dot{\eta} \ , \qquad \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta C_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

In which $\boldsymbol{T_x} = \mathbf{tan}(\boldsymbol{x})$. The matrix $\boldsymbol{W_\eta}$ is invertible if $\theta \neq (2k-1)\phi/2, (k \in \mathbb{Z})$.

The quadcopter is assumed to have symmetric structure with the four arms aligned with the body x- and y-axes. Thus, the inertia matrix is diagonal matrix $\boldsymbol{I}$ in which $\boldsymbol{I_{xx}} = \boldsymbol{I_{yy}}$

Equation 5: Inertia matrix

$$\boldsymbol{I} = \begin{bmatrix} \boldsymbol{I_{xx}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I_{yy}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I_{zz}} \end{bmatrix}$$

The angular velocity of rotor $\boldsymbol{i}$, denoted with $\boldsymbol{\omega_i}$, creates force $\boldsymbol{f_i}$ in the direction of the rotor axis. The angular velocity and acceleration of the rotor also create torque $\boldsymbol{\tau_{M_i}}$ around the rotor axis

Equation 6

$$f_i = k\omega_i^2, \qquad \tau_{M_i} = b\omega_i^2 + I_M\dot{\omega}_\iota,$$

In which the lift constant in $\boldsymbol{k}$, the drag contant is $\boldsymbol{b}$ and the inertia moment of the rotor is $\boldsymbol{I_M}$. Usually the effect of $\boldsymbol{\dot{\omega}_\iota}$ is considered small and thus it is omitted.

The combined forces of rotors create thrust $\boldsymbol{T}$ in the direction of the body z-axis. Torque $\boldsymbol{\tau_B}$ consists of the torques $\boldsymbol{\tau_\phi}$, $\boldsymbol{\tau_\theta}$ and $\boldsymbol{\tau_\psi}$in the direction of the corresponding body frame angles

Equation 7

$$T = \sum_{i=1}^{4} f_i = k\sum_{i=1}^{4} \omega_i^2 \qquad T^B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

$$\tau_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} l\,k\,(-\omega_2^2 + \omega_4^2) \\ l\,k\,(-\omega_1^2 + \omega_4^2) \\ \sum_{i=1}^{4} \tau_{M_i} \end{bmatrix}$$

in which $\boldsymbol{l}$ is the distance between the rotor and the center of mass of the quadcopter. Thus, the roll movement is acquired by decreasing the 2nd rotor velocity and increasing the 4th rotor velocity. Similarly, the pitch movement is acquired by decreasing the 1st rotor velocity and increasing the 3th rotor velocity. Yaw movement is acquired by increasing the the angular velocities of two opposite rotors and decreasing the velocities of the other two.

2.1.1 Newton-Euler Equations

The quadcopter is assumed to be rigid body and thus Newton-Euler equations can be used to describe its dynamics. In the body frame, the force required for the acceleration of mass $m\dot{\boldsymbol{V_B}}$ and the centrifugal force $\mathrm{v} \times (m\boldsymbol{V_B})$ are equal to the gravity $\boldsymbol{R^T G}$ and the total thrust of the rotors $\boldsymbol{T_B}$

Equation 8

$$m\dot{V}_B + \mathrm{v} \times (mV_B) = R^T G + T_B$$

In the inertial frame, the centrifugal force is nullified. Thus, only the gravitational force and the magnitude and direction of the thrust are contributing in the acceleration of the quadcopter

Equation 9

$$m\ddot{\xi} = G + RT_B,$$

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = -g\begin{bmatrix}0\\0\\1\end{bmatrix} + \frac{T}{m}\begin{bmatrix}C_\psi S_\theta C_\phi + S_\psi S_\phi\\ S_\psi S_\theta C_\phi - C_\psi S_\phi\\ C_\theta C_\phi\end{bmatrix}.$$

In the body frame, the angular acceleration of the inertia $\boldsymbol{I}\dot{\mathrm{v}}$, the centripetal forces $\mathrm{v}\times(\boldsymbol{I}\mathrm{v})$ and the gyroscopic forces $\boldsymbol{\Gamma}$ are equal to the external torque $\boldsymbol{\tau}$

Equation 10

$$I\dot{\mathrm{v}} + \mathrm{v}\times(I\mathrm{v}) + \Gamma = \tau$$

$$\dot{\mathrm{v}} = I^{-1}\left(-\begin{bmatrix}p\\q\\r\end{bmatrix}\times\begin{bmatrix}I_{xx}p\\I_{yy}q\\I_{zz}r\end{bmatrix} - I_r\begin{bmatrix}p\\q\\r\end{bmatrix}\times\begin{bmatrix}0\\0\\1\end{bmatrix}\omega_\Gamma + \tau\right)$$

$$\begin{bmatrix}\dot{p}\\\dot{q}\\\dot{r}\end{bmatrix} = \begin{bmatrix}(I_{yy}-I_{zz})qr/I_{xx}\\(I_{zz}-I_{xx})pr/I_{yy}\\(I_{xx}-I_{yy})pq/I_{zz}\end{bmatrix} - I_r\begin{bmatrix}q/I_{xx}\\-p/I_{yy}\\0\end{bmatrix}\omega_\Gamma + \begin{bmatrix}\tau_\phi/I_{xx}\\\tau_\theta/I_{yy}\\\tau_\psi/I_{zz}\end{bmatrix}$$

In which $\boldsymbol{\omega_\Gamma} = \omega_1 - \omega_2 + \omega_3 - \omega_4$. The angular accelerations in the inertial frame are then attracted from the body frame accelerations with the transformation matrix $\boldsymbol{W_\eta^{-1}}$ and its time derivative

Equation 11

$$\ddot{\eta} = \frac{d}{dt}\left(W_\eta^{-1}\mathrm{v}\right) = \frac{d}{dt}\left(W_\eta^{-1}\right)\mathrm{v} + W_\eta^{-1}\dot{\mathrm{v}}$$

$$= \begin{bmatrix}0 & \dot{\phi}C_\phi T_\theta + \dot{\theta}S_\phi/C_\theta^2 & \dot{\phi}S_\phi C_\theta + \dot{\theta}C_\phi/C_\theta^2\\ 0 & -\dot{\phi}S_\phi & -\dot{\phi}C_\phi\\ 0 & \dot{\phi}C_\phi/C_\theta + \dot{\phi}S_\phi T_\theta/C_\theta & -\dot{\phi}S_\phi/C_\theta + \dot{\theta}C_\phi T_\theta/C_\theta\end{bmatrix}\mathrm{v} + W_\eta^{-1}\dot{\mathrm{v}}$$

2.1.2 Aerodynamical Effects

The preceding model is a simplification of complex dynamic interactions. To enforce more realistically behavior of the quadcopter, drag force generated by the air resistance is included. This is devised to Equation 9 and **Erro! Fonte de referência não encontrada.** with the diagonal coefficient matrix associating the linear velocities to the force slowing the movement, as in [55],

Equation 12

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = -g\begin{bmatrix}0\\0\\1\end{bmatrix} + \frac{T}{m}\begin{bmatrix}C_\psi S_\theta C_\phi + S_\psi S_\phi\\ S_\psi S_\theta C_\phi - C_\psi S_\phi\\ C_\theta C_\phi\end{bmatrix} - \frac{1}{m}\begin{bmatrix}A_x & 0 & 0\\ 0 & A_y & 0\\ 0 & 0 & A_z\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix}$$

In which $\boldsymbol{A_x}$, $\boldsymbol{A_y}$ and $\boldsymbol{A_z}$ are the drag force coefficients for velocities in the corresponding directions of the inertial frame.

Several other aerodynamical effects could be included in the model. For example, dependence of thrust on angle of attack, blade flapping and airflow distruptions have been studied in [56] and [57]. The influence of aerodynamical effects are complicated and the effects are difficult to model. Also some of the effects have significant effect only in high velocities. Thus, these effects are excluded from the model and the presented simple model is used.

2.2 Quadcopter Control Theory

Deriving a simplified quadcopter mathematical model makes possible the design of a control system. The inputs to the system consist of the angular velocities of each rotor. Note that in the simplified model they are only the square of the angular velocities, $\boldsymbol{\omega_i^2}$, and never the angular velocity itself, $\boldsymbol{\omega_i}$. For notation simplicity, the inputs will be assumed as $\boldsymbol{\gamma_i} = \boldsymbol{\omega_i^2}$. Since $\boldsymbol{\omega_i}$ can be set, $\boldsymbol{\gamma_i}$ is clearly set as well. With this, now is possible to write the system as a first order differential equation in state space. Let $\boldsymbol{x_1}$ be the position in space of the quadcopter, $\boldsymbol{x_2}$ be the quadcopter linear velocity, $\boldsymbol{x_3}$ be the roll, pitch, and yaw angles, and $\boldsymbol{x_4}$ be the angular velocity vector. (Note that all of these are 3-vectors)

With these being the current state, let's write the state space equations for the evolution of it.

Equation 13

$$\dot{x_1} = x_2$$

$$\dot{x_2} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m} R T_B + \frac{1}{m} F_D$$

$$\dot{x_3} = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & c_\theta s_\phi \\ 0 & -s_\phi & c_\theta c_\phi \end{bmatrix}^{-1} x_4$$

$$\dot{x_4} = \begin{bmatrix} (\tau_\phi I_{xx})^{-1} \\ (\tau_\theta I_{yy})^{-1} \\ (\tau_\psi I_{zz})^{-1} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} \omega_y \omega_z \\ \frac{I_{zz} - I_{xx}}{I_{yy}} \omega_x \omega_z \\ \frac{I_{xx} - I_{yy}}{I_{zz}} \omega_x \omega_y \end{bmatrix}$$

Note that the inputs are not used in these equations directly. However allow us to choose values for $\boldsymbol{\tau}$ and $\boldsymbol{T}$, and then solve for values of $\boldsymbol{\gamma_i}$.

2.2.1 PD Control

This section will bring a brief and simplified explanation about the PD control, how to integrate this control with the RL control will be explained later.

As the main objective is to control the angular velocities and position, the PID control will only be able to use the angle derivatives in the controller; these measured values will give the derivative of the error, and their integral will provide the actual error. For Cartesian position control, the procedures are the same however the dynamic mathematical model refers to the outer loop control (Figure 1.10). For this work, the idea is only to stabilize the quadcopter in a horizontal position, so the desired velocities and angles will all be at zero. Torques are related to the angular velocities by $\boldsymbol{\tau} = \boldsymbol{I\ddot{\theta}}$, so let's set the torques proportional to the output of our controller, with $\boldsymbol{\tau} = \boldsymbol{Iu}(\boldsymbol{t})$. Thus,

Equation 14

$$\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} -I_{xx}\left(K_d\dot{\phi} + K_p\int_0^t \dot{\phi}dt\right) \\ -I_{yy}\left(K_d\dot{\theta} + K_p\int_0^t \dot{\theta}dt\right) \\ -I_{zz}\left(K_d\dot{\psi} + K_p\int_0^t \dot{\psi}dt\right) \end{bmatrix}$$

From the previously derived relationship between torque and inputs,

Equation 15

$$\tau_B = \begin{bmatrix} Lk(\gamma_1 - \gamma_3) \\ Lk(\gamma_2 - \gamma_4) \\ b(\gamma_1 - \gamma_2 + \gamma_3 - \gamma_4) \end{bmatrix} = \begin{bmatrix} -I_{xx}\left(K_d\dot{\phi} + K_p\int_0^t \dot{\phi}dt\right) \\ -I_{yy}\left(K_d\dot{\theta} + K_p\int_0^t \dot{\theta}dt\right) \\ -I_{zz}\left(K_d\dot{\psi} + K_p\int_0^t \dot{\psi}dt\right) \end{bmatrix}$$

This gives a set of three equations with four unknowns. Then, is possible to constrain this: enforcing the constraint that inputs must keep the quadcopter aloft:

Equation 16

$$T = mg$$

Note that this equation ignores the fact that the thrust will not be pointed directly up. This will limit the controller applicability, but should not cause major problems for small deviations from stability. If the gyro sensor is precise enough, it makes possible to integrate the values obtained from the gyro to get the angles $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. In this case, the thrust necessary to keep the quadcopter aloft by projecting the thrust ($\boldsymbol{mg}$) onto the inertial $\boldsymbol{z}$ axis is,

Equation 17

$$T_{proj} = mg\ cos\ \theta \cos\phi$$

Therefore, with a precise angle measurement, the thrust would be equal to

Equation 18

$$T = \frac{mg}{cos\theta\ cos\phi}$$

In which case the component of the thrust pointing along the positive z axis will be equal to ($\boldsymbol{mg}$). Is known that the thrust is proportional to a weighted sum of the inputs:

Equation 19

$$T = \frac{mg}{cos\theta\ cos\phi} = k\sum\gamma_i => \sum\gamma_i = \frac{mg}{k\ cos\theta\ cos\phi}$$

With this extra constraint, now there are a set of four linear equations with four unknowns $\boldsymbol{\gamma_i}$. Solving then for each $\boldsymbol{\gamma_i}$ obtains the following input values:

Equation 20

$$\gamma_1 = \frac{mg}{4k\cos\theta\cos\phi} - \frac{2be_\phi I_{xx} + e_\psi I_{zz} kL}{4bkL}$$

$$\gamma_2 = \frac{mg}{4k\cos\theta\cos\phi} + \frac{e_\psi I_{zz}}{4b} - \frac{e_\theta I_{yy}}{2kL}$$

$$\gamma_3 = \frac{mg}{4k\cos\theta\cos\phi} - \frac{-2be_\phi I_{xx} + e_\psi I_{zz} kL}{4bkL}$$

$$\gamma_4 = \frac{mg}{4k\cos\theta\cos\phi} + \frac{e_\psi I_{zz}}{4b} + \frac{e_\theta I_{yy}}{2kL}$$

2.2.2 PID Control

A PID control is a PD control with another term added, which is proportional to the integral of the process variable. Adding an integral term causes any remaining steady-state error to build up and enact a change, so a PID controller should be able to track the trajectory (and stabilize the quadcopter) with a significantly smaller steady-state error. The equations remain identical to the ones presented in the PD case, but with an additional term in the error:

Equation 21

$$e_\phi = K_d\dot{\phi} + K_p\int_0^t \dot{\phi}dt + K_i\int_0^t\int_0^t \dot{\phi}dt\ dt$$

$$e_\theta = K_d\dot{\theta} + K_p\int_0^t \dot{\theta}dt + K_i\int_0^t\int_0^t \dot{\theta}dt\ dt$$

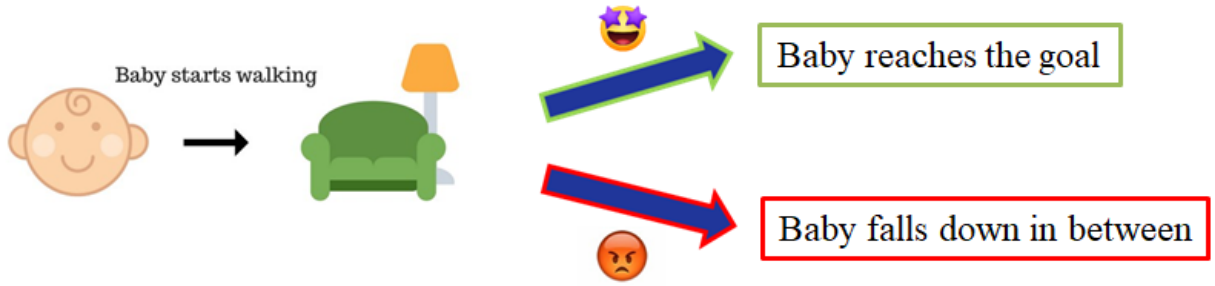
$$e_{\psi} = K_d\dot{\psi} + K_p\int_0^t \dot{\psi}dt + K_i\int_0^t\int_0^t \dot{\psi}dt\,dt$$

2.3 Reinforced Learning

The best definition is given by Barton and Sutton (2015) and reproduced here: “Reinforcement learning is like many topics with names ending in -ing, such as machine learning, planning, and mountaineering. Reinforcement learning problems involve learning what to do, how to map situations to actions, so as to maximize a numerical reward signal. In an essential way they are closed-loop problems because the learning system's actions in sequence its later inputs. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These three characteristics: being closed-loop in an essential way, not having direct instructions as to what actions to take, and where the consequences of actions, including reward signals, play out over extended time periods, are the three most important distinguishing features of reinforcement learning problems.”

In a simple way, let’s consider a cute baby learning to walk and trying to reach his target, a coach. The elements of reinforced learning will be explained in the next item, but the baby body is the environment, what the baby thinks to do is the policy and how good the baby fells after each action is the reward.

Figure 2.2: Reinforced Learning - Example



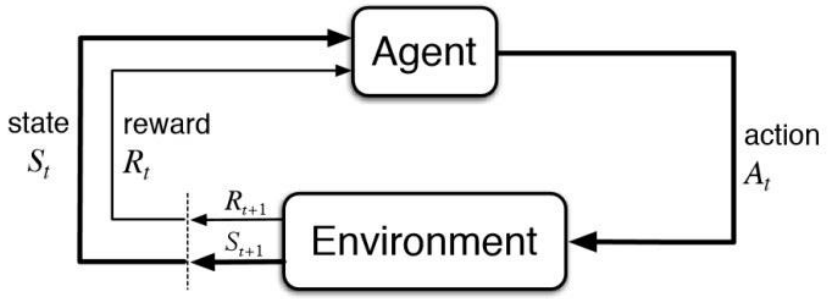
Source: Author

If the baby reaches the goal, he will be happy (high reward) and learns which actions to make that will lead him to the goal. However, if the baby falls down in between, he will be angry (low reward) and learns which actions he shouldn’t take.

2.3.1 Elements of Reinforced Learning

Beyond the agent and the environment, one can identify four main sub elements of a reinforcement learning system: a policy, a reward signal , a value function, and, optionally, a model of the environment.

Figure 2.3: Reinforced Learning - Basic schematic



Source: Barton and Sutton (2015)

A ***policy*** defines how the agent will behave given a state and/or an old action. Thus, is a kind of map that maps states to actions. Talking for engineers, the policy is equivalent to the control logic in a control system. In general, policies may be stochastic, however for specific control tasks, the policy is taken as deterministic.

A ***reward signal*** defines how good or bad the system/environment (quadcopter) was at taking the sent action from the policy. On each time step, the environment sends to the reinforcement learning agent a single number as reward. The agent's main objective is to maximize the total reward. In a biological system, we might think of rewards as analogous to the experiences of pleasure or pain and the reward sent to the agent at any time depends on the agent's current action and the current state of the agent's environment.

A ***value function*** gives the expected reward in a long run, it means, how good is the given state for the future reward. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

The final element is named ***model*** of the environment. Roughly speaking, this is the system to be controlled, it means, the quadcopter. Thus, given a state and action, the model might predict the resultant next state and next reward.

2.4 Reinforcement Learning Algorithms

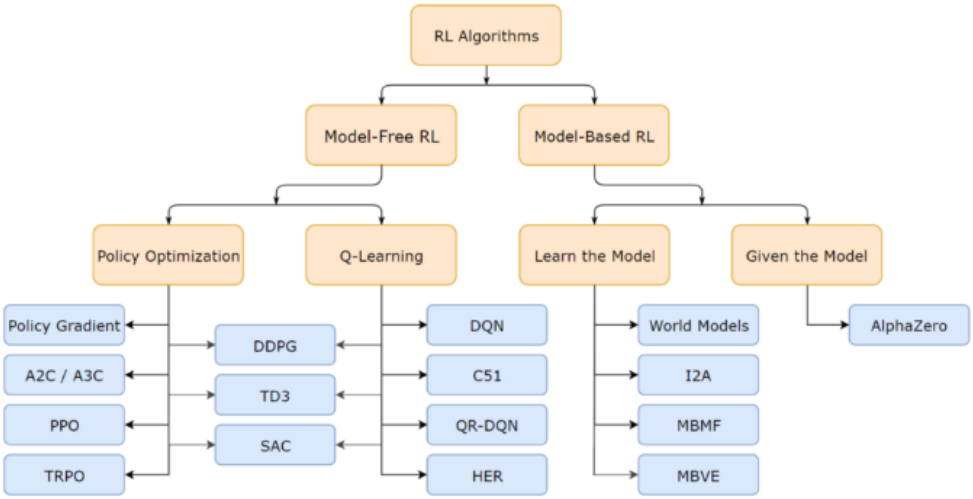
This section pursues to highlight in a non-exhaustive manner the main type of algorithms used for reinforcement learning (RL). The goal is to provide an overview of existing RL methods on an intuitive level by avoiding any deep dive into the models or the math behind it.

Model-free methods are statistically less efficient than model-based methods, because information from the environment is combined with previous, and possibly erroneous, estimates or beliefs about state values, rather than being used directly. [58] On the other hand, uses experience to learn directly

one or both of two simpler quantities (state/ action values or policies), which can achieve the same optimal behavior but without estimation or use of a world model.

Model-based RL uses experience to construct an internal model of the transitions and immediate outcomes in the environment. Appropriate actions are then chosen by searching or planning in this world model.

Figure 2.4: Reinforced Learning taxonomy as defined by openAI



Source: OpenAI (2019)

2.4.1 Model-Free Reinforced Learning

Two main approaches to represent agents with model-free reinforcement learning is Policy optimization and Q-learning.

2.4.1.1 Policy optimization or policy-iteration methods

In policy optimization methods the agent learns directly the policy function that maps state to action. The policy is determined without using a value function.

Important to mention that there are two types of policies: deterministic and stochastic. Deterministic policy maps state to action without uncertainty. It happens when you have a deterministic environment like a chess table. Stochastic policy outputs a probability distribution over actions in a given state. This process is called Partially Observable Markov Decision Process (POMDP)

Policy Gradient (PG)

In this method, the policy $\boldsymbol{\pi}$ has a parameter $\boldsymbol{\theta}$. This $\boldsymbol{\pi}$ outputs a probability distribution of actions.

Equation 22

$$\pi(a|s) = P[a|s]$$

Then is a must to find the best parameters ($\boldsymbol{\theta}$) to maximize (optimize) a score function **J($\boldsymbol{\theta}$)**, given the discount factor $\boldsymbol{\gamma}$ and the reward $\boldsymbol{r}$.

Equation 23

$$J(\theta) = \mathbb{E}_{\pi\theta}[\sum \gamma r]$$

Main steps:

- Measure the quality of a policy with the policy score function.
- Use policy gradient ascent to find the best parameter that improves the policy.

Asynchronous Advantage Actor-Critic (A3C)

This method was published by Google's DeepMind group and covers the following key concept embedded in it is naming:

- **Asynchronous:** Several agents are trained in it is own copy of the environment and the model form these agent's are gathered in a master agent. The reason behind this idea is that the experience of each agent is independent of the experience of the others. In this way the overall experience available for training becomes more diverse.
- **Advantage:** Similarly to PG where the update rule used the discounted returns from a set of experiences in order to tell the agent which actions were "good" or "bad".
- **Actor-critic:** combines the benefits of both approaches from policy-iteration method as PG and value-iteration method as Q-learning (See below). The network will estimate both a value function **V(s)** (how good a certain state is to be in) and a policy **π(s).**

Trust Region Policy Optimization (TRPO)

A on-policy algorithm that can be used or environments with either discrete or continuous action spaces. TRPO updates policies by taking the largest step possible to improve performance, while satisfying a special constraint on how close the new and old policies are allowed to be.

Proximal Policy Optimization (PPO)

Also, an on-policy algorithm which similarly to TRPO can perform on discrete or continuous action spaces. PPO shares motivation with TRPO in the task of answering the question: how to increase policy improvement without the risk of performance collapse? The idea is that PPO improves the stability of the Actor training by limiting the policy update at each training step.

PPO became popular when OpenAI made a breakthrough in Deep RL when they released an algorithm trained to play Dota2 and they won against some of the best players in the world.

2.4.1.2 Q-learning or value-iteration methods

Q-learning learns the action-value function **Q(s, a)**: how good to take an action at a particular state. Basically a scalar value is assigned over an action a given the state **s**.

Deep Q Neural Network (DQN)

DQN is Q-learning with Neural Networks. The motivation behind is simply related to big state space environments where defining a Q-table would be a very complex, challenging and time-consuming task. Instead of a Q-table Neural Networks approximate Q-values for each action based on the state.

Distributional Reinforcement Learning with Quantile Regression (QR-DQN)

In QR-DQN for each state-action pair instead of estimating a single value a distribution of values in learned. The distribution of the values, rather than just the average, can improve the policy. This means that quantiles are learned which threshold values attached to certain probabilities in the cumulative distribution function.

2.4.1.3 Hybrid

Simply as it sounds, these methods combine the strengths of Q-learning and policy gradients, thus the policy function that maps state to action and the action-value function that provides a value for each action is learned. Some hybrid model-free algorithms are:

- Deep Deterministic Policy Gradients (DDPG)
- Soft Actor -Critic (SAC)
- Twin Delayed Deep Deterministic Policy Gradients (TD3)

2.4.2 Model-Base Reinforced Learning

Model-based RL has a strong influence from control theory, and the goal is to plan through an f(s,a) control function to choose the optimal actions. Think it as the RL field where the laws of physics are provided by the creator. The drawback of model-based methods is that although they have more assumptions and approximations on a given task, but may be limited only to these specific types of tasks. There are two main approaches: learning the model or learn given the model.

2.4.2.1 Learn the Model

To learn the model a base policy is ran, like a random or any educated policy, while the trajectory is observed. The model is fitted using the sampled data. Below steps describe the procedure:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

Supervised learning is used to train a model to minimize the least square error from the sampled data for the control function. Optimal trajectory using the model and a cost function is used in step three. The cost function can measure how far we are from the target location and the amount of effort spent.

- **World models:** one of my favorite approaches in which the agent can learn from it's own "dreams" due to the Variable Auto-encoders.
- **Imagination-Augmented Agents (I2A**): learns to interpret predictions from a learned environment model to construct implicit plans in arbitrary ways, by using the predictions as additional context in

deep policy networks. Basically it is a hybrid learning method because it combines model-baes and model-free methods.

- **Model-Based Priors for Model-Free Reinforcement Learning (MBMF**): aims to bridge the gap between model-free and model-based reinforcement learning.
- **Model-Based Value Expansion (MBVE):** this method controls for uncertainty in the model by only allowing imagination to fixed depth. By enabling wider use of learned dynamics models within a model-free reinforcement learning algorithm, we improve value estimation, which, in turn, reduces the sample complexity of learning.

2.4.2.2 Given the Model

This method is becoming famous in recent time due AlphaGo Zero, that defeated the best go player in the world. You can found anything you want on Deep Mind's website.

2.5 Talking about Reward

This section will give an explanation about reward methods. The notation here may have some similar symbols with the quadcopter dynamics or even control; however they are NOT related to each other.

A reinforced learning algorithm can be used to solve problems modelled as Markov decision processes (MDPs). An MDP is a tuple $< \boldsymbol{X}, \boldsymbol{U}, \boldsymbol{f}, \boldsymbol{\rho} >$, where $\boldsymbol{X}$ denotates the state space, $\boldsymbol{U}$ the action space, $\boldsymbol{f} : X \times U \times X \to [0, \infty)$ the state transition probability density function and $\boldsymbol{\rho}: X \times U \times X \to \mathbb{R}$ the reward function.

It is important to note that since state space is continuous, it is only possible to define a probability of reaching a certain state *region*, since the probability of reaching a particular state is zero. So, assuming that a stochastic process to be controlled can be described by the state transition probability density function $\boldsymbol{f}$. The probability of reaching a state $\boldsymbol{x_{k+1}}$ in the region $\boldsymbol{X_{k+1}} \subseteq \boldsymbol{X}$ from state $\boldsymbol{x_k}$ after applying action $\boldsymbol{u_k}$ is

Equation 24

$$P(x_{k+1} \in X_{k+1} | x_k, u_k) = \int_{X_{k+1}} f(x_k, u_k, x') dx'$$

After each transition to a state $\boldsymbol{x_{k+1}}$, the controller receives an immediate reward

Equation 25

$$r_{k+1} = \rho(x_k, u_k, x_{k+1})$$

which depends on the previous state, the current state and the action taken. The action $\boldsymbol{u_k}$ taken in a state $\boldsymbol{x_k}$ is drawn from a stochastic policy $\boldsymbol{\pi}: X \times U \to [0, \infty)$.

The goal of the reinforced learning agent is to find the policy $\boldsymbol{\pi}$ which maximizes the expected value of a certain function $\boldsymbol{g}$ of the immediate rewards received, while following the policy $\boldsymbol{\pi}$. This expected value is cost-to-go function

Equation 26

$$J(\pi) = \mathbb{E}\{g(r_1, r_2, \dots)|\pi\}$$

In most cases, the function $\boldsymbol{g}$ is either the discounted sum of rewards received, as explained in the next items.

2.5.1 Discounted Reward

In the discounted reward setting, the cost function **J** is equal to the expected value of the discounted sum of rewards when starting from an initial state $\boldsymbol{x_0} \in \boldsymbol{X}$, draw from an initial state distribution $\boldsymbol{x_0} \sim \boldsymbol{d_0}(\cdot)$, also called the discounted return

Equation 27

$$J(\pi) = \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} | d_0, \pi\right\} = \int_X d_\gamma^\pi(x) \int_U \pi(x,u) \int_X f(x,u,x')\rho(x,u,x')dx'du\,dx$$

Where $d_\gamma^\pi(x) = \sum_{k=0}^{\infty} \gamma^k p(x_k = x|d_0, \pi)$ is the discounted state distribution under the policy $\boldsymbol{\pi}$ and $\boldsymbol{\gamma} \in [\mathbf{0}, \mathbf{1})$ denotes the reward discount factor.

During the learning process, the agent will have to estimate the cost-to-go function $\boldsymbol{J}$ for a given policy $\boldsymbol{\pi}$. This procedure is called *policy evaluation.* The resulting estimate of $\boldsymbol{J}$ is called the *value function* and two definitions exists for it. The **state value** function

Equation 28

$$V^\pi(x) = \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} |\, x_0 = x, \pi\right\}$$

Only depends on the state $\boldsymbol{x}$ and assumes that the policy $\boldsymbol{\pi}$ is followed starting from this state. The **state-action value** function

Equation 29

$$Q^\pi(x,u) = \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1} | x_0 = x, u_0 = u, \pi\right\}$$

Also depends on the state $\boldsymbol{x}$, but makes action $\boldsymbol{u}$ chosen in this state a free variable instead of having it generated by the policy $\boldsymbol{\pi}$. Once the first transition onto a next state has been made, $\boldsymbol{\pi}$ governs the rest of the action selection. The relationship between two definitions for the value function is given by

Equation 30

$$V^\pi(x) = \mathbb{E}\{Q^\pi(x,u)|\, u \sim \pi(x,\cdot)\}$$

With some manipulation, Equation 28 and Equation 29 can be put into a recursive form. For the state value function this is

Equation 31

$$V^{\pi}(x) = \mathbb{E}\{\rho(x,u,x') + \gamma V^{\pi}(x')\}$$

With $\boldsymbol{u}$ drawn from the probability distribution function $\boldsymbol{\pi}(\boldsymbol{x}, \cdot)$ and $\boldsymbol{x}'$ drawn from $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \cdot)$. For the state-action value function the recursive form is

Equation 32

$$Q^{\pi}(x,u) = \mathbb{E}\{\rho(x,u,x') + \gamma Q^{\pi}(x',u')\}$$

with $\boldsymbol{x}'$ drawn from the probability distribution function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \cdot)$ and $\boldsymbol{u}'$ drawn from the distribution $\boldsymbol{\pi}(\boldsymbol{x}', \cdot)$. These recursive relationships are called ***Bellman equations***.

2.5.2 Average Reward

As an alternative to the discounted reward setting, is the approach of using the *average* return. In this setting a starting state $\boldsymbol{x_0}$ does not need to be chosen, under the assumption that the process is ergotic and, thus $\boldsymbol{J}$ does not depend on the starting state. Instead, the value functions for a policy $\boldsymbol{\pi}$ are defined relative to the average expected reward per time step under the policy, turning the cost-to-go function into

Equation 33

$$J(\pi) = \lim_{n\to\infty}\left(\frac{1}{n}\right)\mathbb{E}\left\{\sum_{k=0}^{n-1} r_{k+1}|\pi\right\} = \int_X d^{\pi}(x)\int_U \pi(x,u)\int_X f(x,u,x')\rho(x,u,x')dx'du\,dx$$

Equation 33 is very similar to Equation 27, except that the definition for the state distribution changed to $d^{\pi}(x) = \lim_{k\to\infty} p(x_k = x, \pi)$. For a given policy $\boldsymbol{\pi}$, the state value function $\boldsymbol{V^{\pi}(x)}$ and state-action value $\boldsymbol{Q^{\pi}(x,u)}$ are then defined as

Equation 34

$$V^{\pi}(x) = \mathbb{E}\left\{\sum_{k=0}^{\infty}\big(r_{k+1} - J(\pi)\big)|x_0 = x, \pi\right\}$$

$$Q^{\pi}(x,u) = \mathbb{E}\left\{\sum_{k=0}^{\infty}\big(r_{k+1} - J(\pi)\big)|x_0 = x, u_0 = u, \pi\right\}$$

The Bellman equations for the average reward – in this case also called the Poisson equations – are

Equation 35

$$V^{\pi}(x) + J(\pi) = \mathbb{E}\{\rho(x,u,x') + V^{\pi}(x')\}$$

With $\boldsymbol{u}$ and $\boldsymbol{x}'$ drawn from the appropriate distributions as before and

Equation 36

$$Q^{\pi}(x,u) + J(\pi) = \mathbb{E}\{\, \rho(x,u,x') + Q^{\pi}(x',u')\}$$

Again with $\boldsymbol{x}'$ and $\boldsymbol{u}'$ drawn from the appropriate distributions.

2.6 Talking about Stochastic Policy Gradient Theorems

Many actor-critic algorithms, now, rely on the policy gradient theorem, proving that an unbiased estimate of the gradient (Equation 46) can be obtained from experience using an approximate value function that satisfies certain properties. Roughly speaking, the basic idea is that since the number of parameters that the actor has to update is relatively small compared to the (usually infinite) number of states, it is not useful to have the critic attempting to compute the exact value function, which is also a high-dimensional object. Instead, it should compute a projection of the value function onto a low-dimensional subspace.

In the case of an approximated stochastic policy, but exact state-action value function $\boldsymbol{Q^{\pi}}$, the policy gradient theorem is as follows.

2.6.1 Theorem 1 (Policy Gradient)

For any MDP, in either the average reward or discounted reward setting, the policy gradient is given by

Equation 37

$$\nabla_{\vartheta} J = \int_{X} d^{\pi}(x) \int_{U} \nabla_{\vartheta} \pi(x,u) Q^{\pi}(x,u) du dx$$

With $\boldsymbol{d^{\pi}(x)}$ defined for the appropriated reward setting.

This clearly shows the relationship between the policy gradient $\boldsymbol{\nabla_{\vartheta} J}$ and the critic function $\boldsymbol{Q^{\pi}(x,u)}$ and ties together the update equations of the actor and critic in the **Erro! Fonte de referência não encontrada.**.

For most applications, the state-action space is continuous and thus infinite, which means that it is necessary to approximate the state or state-action value function. The result shows that $\boldsymbol{Q^{\pi}(x,u)}$ can be approximated with $\boldsymbol{h_w} : X \times U \to \mathbb{R}$, parametrized by $\boldsymbol{w}$, without affecting the unbiasedness of the policy gradient estimate.

In order to find the closest approximation of $\boldsymbol{Q^{\pi}}$ by $\boldsymbol{h_w}$, let's try to find the $\boldsymbol{w}$ that minimizes the quadratic error

Equation 38

$$\varepsilon_{w}^{\pi}\,(x,u) = \frac{1}{2}[Q^{\pi}(x,u) - h_{w}(x,u)]^2$$

The gradient of this quadratic error with respect to $\boldsymbol{w}$ is

Equation 39

$$\nabla_w \varepsilon_w^{\pi}(x,u) = [Q^{\pi}(x,u) - h_w(x,u)]\, \nabla_w h_w(x,u)$$

and this can be used in a gradient descent algorithm to find the optimal $\boldsymbol{w}$. If the estimator of $\boldsymbol{Q^{\pi}(x,u)}$ is unbiased, the expected value of the above equation is zero for the optimal $\boldsymbol{w}$, it means:

Equation 40

$$\int_X d^{\pi}(x) \int_U \pi(x,u) \nabla_w \varepsilon_w^{\pi}(x,u) du dx = 0$$

The policy gradient theorem with function approximation is based on the last equality (Equation 40).

2.6.2 Theorem 2 (Policy Gradient with Function Approximation)

More details about function approximator, for this case a MLP neural network, will be given later in a specific item, those concepts are not necessary to understand the following equations. So, if $\boldsymbol{h_w}$ satisfies Equation 40 and:

Equation 41

$$\nabla_w h_w(x,u) = \nabla_{\vartheta} \ln \pi_{\vartheta}(x,u)$$

where $\boldsymbol{\pi_{\vartheta}(x,u)}$ denotes the stochastic policy, parametrized by $\boldsymbol{\vartheta}$, then

Equation 42

$$\nabla_{\vartheta} J = \int_X d^{\pi}(x) \int_U \nabla_{\vartheta}\, \pi(x,u) h_w(x,u) du dx$$

An extra assumption is that $\boldsymbol{h}$ actually needs to be an approximator that is linear with respect to some parameter $\boldsymbol{w}$ and features $\boldsymbol{\psi}$, i.e. $\boldsymbol{h_w} = w^T \psi(x,u)$, transforming Equation 41 into

Equation 43

$$\psi(x,u) = \nabla_{\vartheta} \ln \pi_{\vartheta}(x,u)$$

Features $\boldsymbol{\psi}$ that satisfy this last equation as known as *compatible* features.

2.7 Talking about Deterministic Policy Gradients

We now consider how the policy gradient framework may be extended to deterministic policies. According to [65], the main result from this item will be a deterministic policy gradient theorem, analogous to the stochastic policy gradient theorem presented in the previous section.

It was previously believed that the deterministic policy gradient did not exist, or could only be obtained when using a model. However, [65] shows that the deterministic policy gradient does indeed exist, and

furthermore it has a simple model-free form that simply follows the gradient of the action-value function. In addition, it also shows that deterministic policy gradient is the limiting case, as policy variance tends to zero, of the stochastic policy gradient.

The majority of model-free reinforced learning algorithms are based on generalized policy iteration: interleaving ***policy evaluation*** with ***policy improvement***. Policy evaluation methods estimate the action-value function $\boldsymbol{Q^{\pi}(s,a)}$ or $\boldsymbol{Q^{w}(s,a) = h_{w}(s,a)}$, for example by Monte-Carlo evaluation or temporal-difference learning (TD). Policy improvement methods update the policy with respect to the (estimated) action-value function.

A simple and computationally attractive method is to move the policy in the direction of the gradient of $\boldsymbol{Q}$. Specifically, for each visited state s, the policy parameter $\boldsymbol{\vartheta^{k+1}}$ are updated in proportion to the gradient $\boldsymbol{\nabla_{\vartheta} Q^{w^{k}}(s, \pi_{\vartheta}(s))}$.

Equation 44

$$\nabla_{\vartheta} J(\pi_{\vartheta}) = \mathbb{E}\{\nabla_{\vartheta} \pi_{\vartheta}(s)\, \nabla_{a} Q^{w}(s,a) |\, a = \pi_{\vartheta}(s)\}$$

2.8 Actor-Critic Reinforced Learning

This section will give an explanation on all three groups, starting with critic-only methods. The notation here may have some similar symbols with the quadcopter dynamics or even control, however they are NOT related to each other. The part on actor-only methods introduces the concept of a policy gradient, which provides the basis for actor-critic algorithms. The final part of this section explains the policy gradient theorem, an important result that is now widely used in many implementations of actor-critic algorithms.

In real-life applications, such as robotics, processes usually have continuous state and action spaces, making it impossible to store exact value functions or policies for each separate state or state-action pair. Any RL algorithm used in practice will have to make use of function approximators for the value function and/or the policy in order to cover the full range of states and actions. Therefore, this section assumes the use of such function approximators.

2.8.1 Critic-Only Methods

Critic-only methods, such as Q-learning and SARSA, use a state-action value function and no explicit function for the policy. For continuous state and action spaces, this will be an approximate state-action value function. These methods learn the optimal value function by finding online an approximate solution to the Bellman equation. A deterministic policy, denoted by $\boldsymbol{\pi}: \boldsymbol{X} \rightarrow \boldsymbol{U}$ is calculated by using an optimization procedure over the value function

Equation 45

$$\pi(x) = arg \max_{u} Q(x,u)$$

There is no reliable guarantee on the near-optimality of the resulting policy for just any approximated value function when learning in an online setting. For example, Q-learning and SARSA with specific function approximators have been shown not to converge even for simple MDPs. However, was shown that convergence can be assured for linear-in-parameters function approximators if trajectories are sampled according to their on-policy distribution. Nevertheless, for most choices of basis functions an approximated value function learned by temporal difference learning will be biased. This is reflected by the state-of-the-art bounds on the least-squares temporal difference (LSTD) solution quality, which always include a term depending on the distance between the true value function and its projection on the approximation space. For a particularly bad choice of basis functions, this bias can grow very large.

2.8.2 Actor-only Methods and the Policy Gradient

Policy gradient methods (see, for instance, the SRV [59] and Williams' REINFORCE algorithms [60]) are principally actor-only and do not use any form of stored value function. Instead, the majority of actor-only algorithms work with a parametrized family of policies and optimize the cost defined directly over the parameter scape of the policy. A major advantage of actor-only methods over critic-only methods is that they allow the policy to generate actions in the complete continuous action space.

A policy gradient method is generally obtained by parametrizing the policy $\boldsymbol{\pi}$ by the parameter vector $\boldsymbol{\vartheta} \in \mathbb{R}^{\boldsymbol{p}}$. Considering that both Equation 27 and Equation 33 are functions of the parametrized policy $\boldsymbol{\pi_\vartheta}$, they are in fact functions of $\boldsymbol{\vartheta}$, the gradient of the cost function with respect to $\boldsymbol{\vartheta}$ is described by

Equation 46

$$\nabla_\vartheta J = \frac{\partial J}{\partial \pi_\vartheta} \frac{\partial \pi_\vartheta}{\partial \vartheta}$$

Then, by using standard optimization techniques, a locally optimal solution of the cost $\boldsymbol{J}$ can be found. The gradient $\boldsymbol{\nabla_\vartheta J}$ is estimated per time step and the parameters are then updated in the direction of this gradient. For example, a simple gradient ascent method would yield the policy gradient update equation

Equation 47

$$\vartheta_{k+1} = \vartheta_k + \alpha_{a,k} \nabla_\vartheta J_k$$

where $\boldsymbol{\alpha_{a,k} > 0}$ is a small enough learning rate for the actor by which it is obtained that $\boldsymbol{J(\vartheta_{k+1}) \geq J(\vartheta_k)}$.

The main advantage of the actor-only approach is their strong convergence property, which naturally inherited from gradient descent methods. Convergence is obtained if the estimated gradients are unbiased and the learning rates $\boldsymbol{\alpha_{a,k}}$ satisfy,

Equation 48

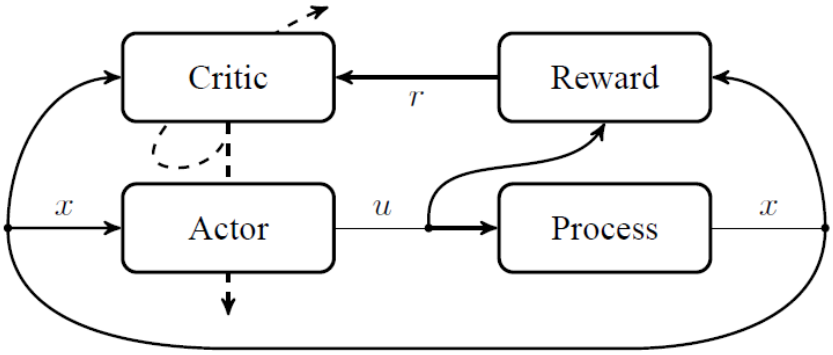
$$\sum_{k=0}^{\infty} \alpha_{a,k} = \infty \qquad \sum_{k=0}^{\infty} \alpha_{a,k}^2 < \infty$$

A drawback of the actor-only approach is that the estimated gradient may have a large variance. Also, every gradient is calculated without using any knowledge of the past estimates.

2.8.3 Actor-Critic Algorithms – Stochastic

Actor-critic methods aim to combine the advantages of actor-only and critic-only methods. Like actor-only methods, actor-critic methods are capable of producing continuous actions, while the large variance in the policy gradients of actor-only methods is countered by adding a critic. The role of the critic is to evaluate the current policy prescribed by the actor. In principle, this evaluation can be done by any policy evaluation method commonly used, such as **TD(λ), LSTD**, or **residual gradients**. The critic approximates and updates the value function using samples. The value function is then used to update the actor's policy parameters in the direction of performance improvement. These methods usually preserve the desirable convergence properties of policy gradient methods, in contrast to critic-only methods. In actor-critic methods, the policy is not directly inferred from the value function by using $\boldsymbol{\pi(x)} = arg\max_u Q(x,u)$. Instead, the policy is updated in the policy gradient direction using only a small step size $\boldsymbol{\alpha_a}$, meaning that a change in the value function will only result in a small change in the policy, leading to less or no oscillatory behavior in the policy as described in [61].

Figure 2.5: Schematic overview of an actor-critic algorithm.



Source: Grondman, Ivo et Al. (2012)

Figure 2.5 shows the schematic structure of an actor-critic algorithm. The learning agent has been split into two separate entities: the actor (policy) and the critic (value function). The actor is only responsible for generating a control input $\boldsymbol{u}$, given the current state $\boldsymbol{x}$. The critic is responsible for processing the rewards it receives, i.e. evaluating the quality of the current policy by adapting the value function estimate. After a number of policy evaluation steps by the critic, the actor is updated by using information from the critic.

A unified notion for the actor-critic algorithms described here will be adopted. Remembering: The notation here may have some similar symbols with the quadcopter dynamics or even control, however they are **NOT** related to each other. Therefore, two actor-critic algorithm templates are introduced: one for the discounted reward setting and one for the average reward setting. Once these templates are established, specific actor-critic algorithms can be discussed by only looking at how they fit into the general template or in what way they differ from it.

For both rewards settings, the value function is parametrized by the parameter vector $\theta \in \mathbb{R}^q$. This will be denoted with $\boldsymbol{V_\theta(x)}$ or $\boldsymbol{Q_\theta(x,u)}$. If the parameterization is linear, the features (basis functions) will be denoted with $\boldsymbol{\phi}$, i.e.

Equation 49

$$V_\theta(x) = \theta^T \phi(x) \text{ or } Q_\theta(x,u) = \theta^T \phi(x,u)$$

The stochastic policy $\boldsymbol{\pi}$ is parameterized by $\boldsymbol{\vartheta} \in \mathbb{R}^{\boldsymbol{p}}$ and will be denoted with $\boldsymbol{\pi_\vartheta(x,u)}$. If the policy is denoted with $\boldsymbol{\pi_\vartheta(x)}$, it is deterministic and no longer represents a probability density function, but the direct mapping from states to actions $\boldsymbol{u = \pi_\vartheta(x)}$.

The goal in actor-critic algorithms – or any other RL algorithm for that matter – is to find the best policy possible, given some stationary MDP (Markov Decision Process). A prerequisite for this is that the critic is able to accurately evaluate a given policy. The difference between the right-hand and left-hand side of the Bellman equation, whether it is the one for the discounted reward setting or the average reward setting, is called *Temporal Difference (TD)* error and is used to update the critic. Using the function approximation for the critic and a transition sample $\boldsymbol{(x_k, u_k, r_{k+1}, x_{k+1})}$, the TD error is estimated as

Equation 50

$$\delta_k = r_{k+1} + \gamma V_{\theta_k}(x_{K+1}) - V_{\theta_k}(x_k)$$

Perhaps the most standard way of updating the critic, is to exploit this TD error for use in a gradient descent update:

Equation 51

$$\theta_{k+1} = \theta_k + \alpha_{c,k} \delta_k \nabla_\theta V_{\theta_k}(x_k)$$

where $\boldsymbol{\alpha_{c,k} > 0}$ is the learning rate of the critic. For the linearly parameterized function approximator, this reduces to:

Equation 52

$$\theta_{k+1} = \theta_k + \alpha_{c,k} \delta_k \phi(x_k)$$

This temporal difference method is also known as $\boldsymbol{TD(0)}$ learning, as no eligibility traces are used. The extension to the use of eligibility traces, resulting in $\boldsymbol{TD(\lambda)}$ methods, is straightforward and is explained next.

Using the critic updated formula $\theta_{k+1} = \theta_k + \alpha_{c,k} \delta_k \nabla_\theta V_{\theta_k}(x_k)$, as already shown above, to update the critic results in a one-step backup, whereas the reward received is often the result of a series of steps.

2.8.4 Actor-Critic Algorithms – Deterministic

We consider a standard reinforced learning setup consisting of an agent interacting with an environment in discrete timesteps. At each timestep $\boldsymbol{dt}$ the agent receives an observation $\boldsymbol{x_k}$, takes an action $\boldsymbol{u_k}$ and receives a scalar reward $\boldsymbol{r_k}$. In all the environments considered here the actions are real-valued $\boldsymbol{u_k} \in \mathbb{R}^n$. In general, the environment may be partially observed so that the entire history of the observation, action pair $\boldsymbol{s_k} = (x_1, u_1, \dots, u_{k-1}, x_k)$ may be required to describe the state. Here, we assumed the environment is fully-observed so $\boldsymbol{s_k} = \boldsymbol{x_k}$.

The DDPG algorithm maintains a parametrized actor function $\boldsymbol{\pi}(\boldsymbol{s}|\boldsymbol{\theta^\pi})$ which specifies the current policy by deterministically mapping states to a specific action. The critic $\boldsymbol{Q}(\boldsymbol{x}, \boldsymbol{u})$ is learned using the Bellman equation. The actor is updated by following and applying the chain rule to the expected return from the start distribution $\boldsymbol{J}$ with respect to the actor parameters:

Equation 53

$$\nabla_{\theta^\pi} J \approx \mathbb{E}\{\nabla_{\theta^\pi} Q(x, u|\theta^Q)|\, x = x_k, u = \pi(x_k|\theta^\pi)\}$$

$$= \mathbb{E}\big\{\nabla_u Q(x, u|\theta^Q)\big|\, x = x_k, u = \pi(x_k) \;\times\; \nabla_{\theta_\pi} \pi(x|\theta^\pi)\big|x = x_k\big\}$$

[65] proved that this is the gradient of the policy's performance.

One challenge when using neural networks for reinforcement learning is that most optimization algorithms assume that the samples are independently and identically distributed. Obviously, when the samples are generated from exploring sequentially in an environment this assumption no longer holds. Additionally, to make efficient use of hardware optimizations, it is essential to learn in mini-batches, rather than online.

2.9 Neural Network – Function Approximator

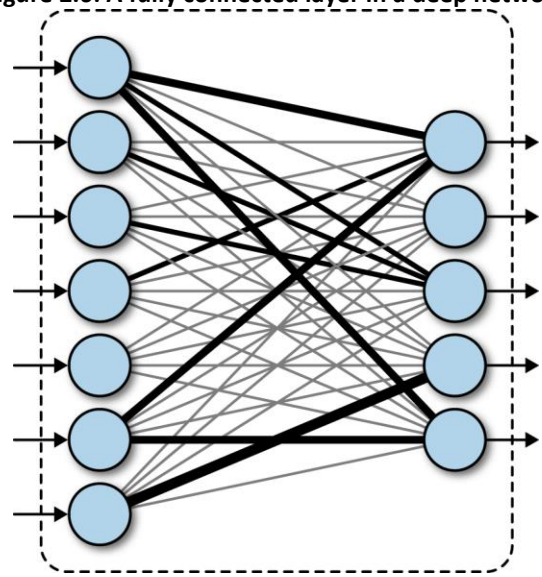
This chapter will introduce you to fully connected deep networks, known as Multi-Layer Perceptron (MLP). MLPs are used for thousands of applications and thus are the workhorses of deep learning. The major advantage of fully connected networks is that they are "structure agnostic." That is, no special assumptions need to be made about the input (for example, that the input consists of images or videos). We will make use of this generality to use fully connected deep networks to address the problem of quadcopter control.

While being structure agnostic makes fully connected networks very broadly applicable, such networks do tend to have weaker performance than special-purpose networks tuned to the structure of a problem space. However, for this work, it will be more than sufficient.

2.9.1 What is a Fully Connected Deep Network?

Exactly as described by O'Reilly (2019), fully connected neural network consists of a series of fully connected layers. A fully connected layer is a function from $\mathbb{R}_m$ to $\mathbb{R}_n$. Each output dimension depends on each input dimension.

Figure 2.6: A fully connected layer in a deep network



Source: O'Reilly (2019)

Let $\boldsymbol{x} \in \mathbb{R}_m$ represents the input to a fully connected layer. Let $\boldsymbol{y_i} \in \mathbb{R}$ be the **i** –th output from the fully connected layer. Then $\boldsymbol{y_i} \in \mathbb{R}$ can be computed as follows:

Equation 54

$$y_i = \sigma(w_1 x_1 + \cdots + w_m x_m)$$

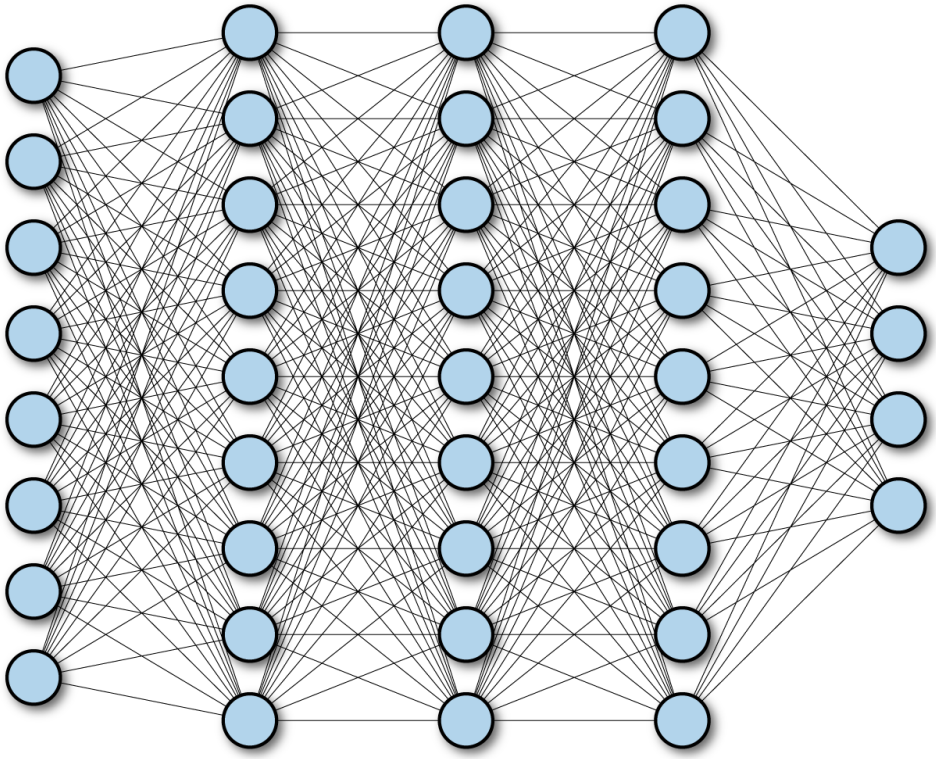
Here, $\boldsymbol{\sigma}$ is the activation function of that neuron, and the $\boldsymbol{w_i}$ are learnable parameters in the network. The full output $\boldsymbol{y}$ is then:

Equation 55

$$y = \sigma\left(w_{1,1} x_1 + \cdots + w_{1,m} x_m\right) \vdots \sigma(w_{n,1} x_1 + \cdots + w_{n,m} x_m)$$

Note that is directly possible to stack fully connected networks. A network with multiple fully connected networks is often called a "deep" network as shown bellow:

Figure 2.7: A multilayer deep fully connected network



Source: O'Reilly (2019)

As a quick implementation note, note that the equation for a single neuron looks very similar to a dot-product of two vectors (recall the discussion of tensor basics). For a layer of neurons, it is often convenient for efficiency purposes to compute $\boldsymbol{y}$ as a matrix multiply:

Equation 56

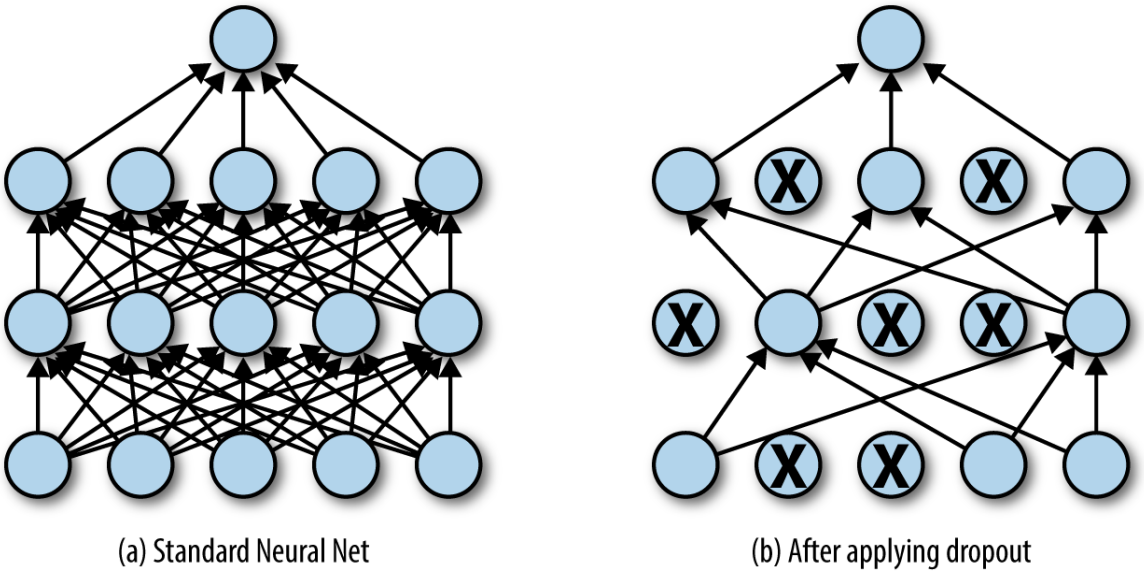
$$y = \sigma(w\ x)$$

where sigma is a matrix in $\mathbb{R}_{n\times m}$ and the nonlinearity $\boldsymbol{\sigma}$ is applied componentwise.

2.9.2 Dropout Regularization

Dropout is a form of regularization that randomly drops some proportion of the nodes that feed into a fully connected layer. Here, dropping a node means that its contribution to the corresponding activation function is set to 0. Since there is no activation contribution, the gradients for dropped nodes drop to zero as well.

Figure 2.8: Dropout randomly drops neurons from a network while training. Empirically, this technique often provides powerful regularization for network training.



Source: O'Reilly (2019)

The nodes to be dropped are chosen at random during each step of gradient descent. The underlying design principle is that the network will be forced to avoid "co-adaptation". Dropout prevents this type of co-adaptation because it will no longer be possible to depend on the presence of single powerful neurons (since that neuron might drop randomly during training). As a result, other neurons will be forced to "pick up the slack" and learn useful representations as well. The theoretical argument follows that this process should result in stronger learned models [64].

3. Methodology

Now we have all the necessary background to make this work done, however, how everything works together? First of all, let's remember the general structure and after that, we will go deeper in each step.

So, as explained in the introduction, we are using DDPG Actor-Critic Reinforced Learning. The RL control system will be dynamic changing the PID constants while trying to stabilize the quadcopter. The actor and critic will be 2 different fully connected neural networks with the same input vector. The actor and critic will be 2 different fully dense connected neural networks with 2 hidden layers. Results will be compared over simple PID control tuned using parameter grid search method to selecting the gains.

3.1 Quadcopter Simulation – RL Environment

The mathematical model of the quadcopter is implemented for simulation in Python 3.X. The parameters used are from the DJI Phantom 2 quadcopter [62].

Table 1: DJI Phantom 2 - Parameters

Variable	Value	Units
K_V	920	[rpm/V]
K_E	$9.5493/K_V$	[V.s/rad]
T_f	0.04	[N.m]
D_f	0.0002	[N.m.s/rad]
r	0.12	[m]
$J_m = I_m$	0.0000049	[Kg.m²]
$\omega_{\max}$	1047.197	[rad/s]
n_B	2	----
m_B	0.0055	[Kg]
r_B	0.12	[m]
ϵ	0.004	[m]
C_T	0.0048	----
C_Q	0.00023515	----
r_{rot}	0.014	[m]
ρ	1.225	[Kg/m³]
m	1.3	[Kg]
l	0.0175	[m]
I_x	0.081	[Kg.m²]
I_y	0.081	[Kg.m²]
I_z	0.142	[Kg.m²]
m_{rot}	0.025	[Kg]

For calculating the parameters $\boldsymbol{k}$ and $\boldsymbol{b}$ from the Quadcopter Mathematical Modelling item, the following equations will be needed:

Equation 57

$$k = C_T \rho A\, r^2$$

$$b = C_Q \rho A r^3$$

where $\boldsymbol{r}$ and $\boldsymbol{A = \pi r^2}$ are the radius and disk area of the propeller, respectively. So, $\boldsymbol{k} = 3.83 \times 10^{-6}$ and $\boldsymbol{b} = 2.25 \times 10^{-8}$

The selected step-size is $\boldsymbol{dt}$ **= 0.005 [s]**, each episode has 30 seconds and there are a total of 1000 episodes with random angular velocities and angular position initialization.

3.2 RL Reward method

The step-size $\boldsymbol{dt}$ executes a single simulation step with the specified actions and returns to the agent the new state vector, together with a reward indicating how well the given action was performed. Reward engineering can be challenging. If careful design is not performed, the derived policy may not reflect what was originally intended. For this work, with the goal of establishing a baseline of accuracy, will be used a reward to reflect the total quadcopter position error.

Translating the current error $\boldsymbol{e_t}$ at time $\boldsymbol{t}$ into a derived reward $\boldsymbol{r_{k+1}}$ as follows [63],

Equation 58: Reward

$$r_{k+1} = -clip(4 \times 10^{-3}\|p_k\| + 2 \times 10^{-4}\|a_k\| + 3 \times 10^{-4}\|\omega_k\| + 5 \times 10^{-4}\|o_k\|)$$

where $\boldsymbol{p_k}$, $\boldsymbol{a_k}$, $\boldsymbol{\omega_k}$ and $\boldsymbol{o_k}$ are error in position, linear acceleration, angular velocities and angular position (Euler angles) respectively. The *clip* function clips the result between the $[0,1]$ in cases where there is an overflow in the error.

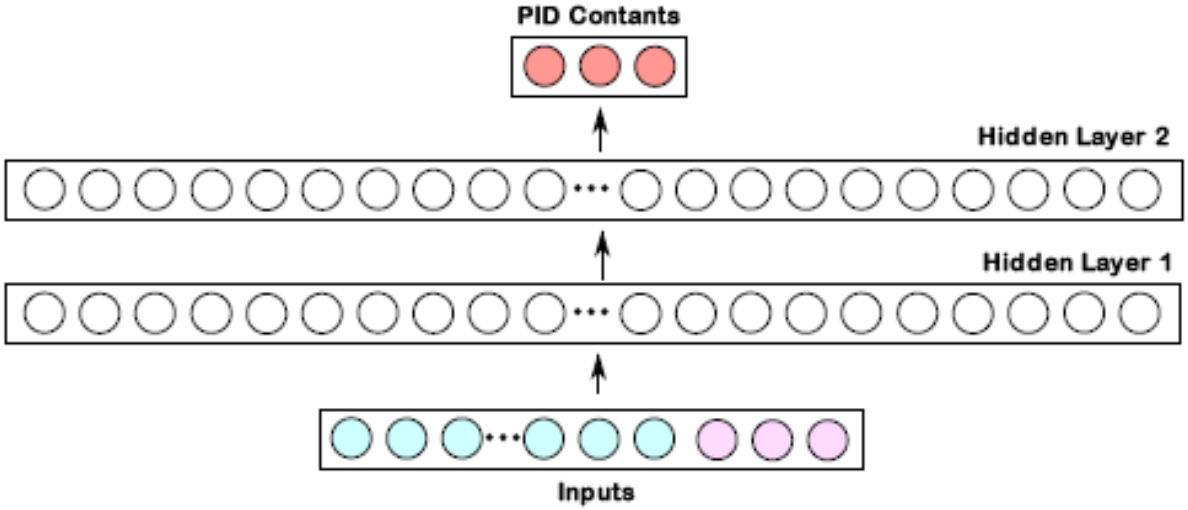
Since the reward is negative, it signifies a penalty, the agent maximizes the reward (and thus minimizing error) overtime in order to track the target as accurately as possible. Rewards are normalized between to provide standardization and stabilization during training.

3.3 Actor and Critic – Structure and Training

Let's first talk about how to build the Actor Network. Here was used 2 'ReLU' hidden layers with 50 hidden units each. The output consist of 3 continuous actions with 'Linear' activation representing each controller constant $[\boldsymbol{K_p}, \boldsymbol{K_i}, \boldsymbol{K_d}]$. The structure is not optimized in any sense. Different number of nodes and activation functions were tried, however hyper parameters were not deeply studied and changed.

In the final layer was used the normal initialization with $\mu = 0$, $\sigma = 1 \times 10^{-4}$ to ensure the initial outputs for the policy were near zero.

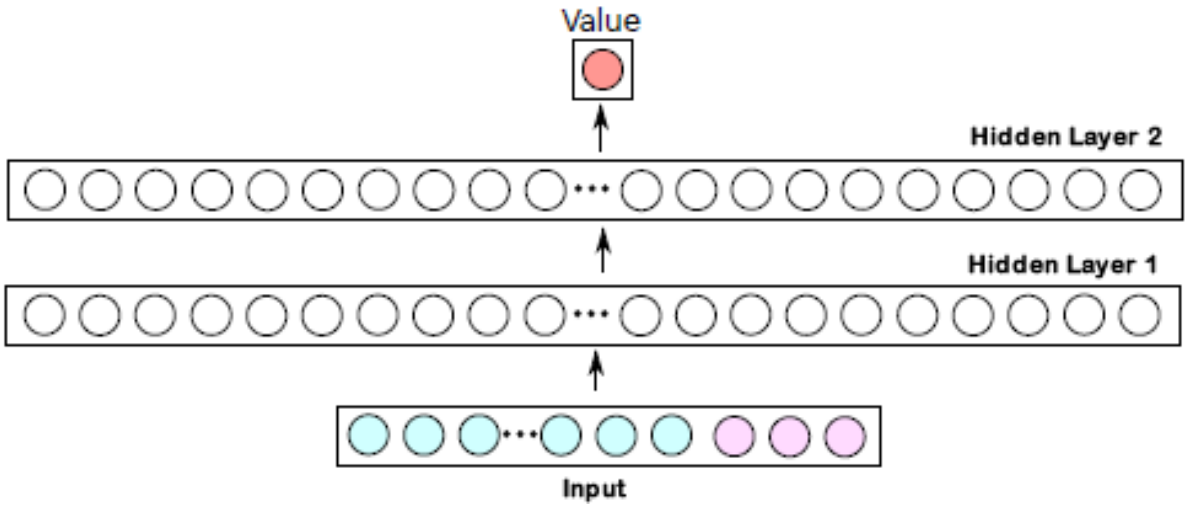
Figure 3.1: Actor Neural Network architecture



Source: Author

The construction of the Critic Network is very similar to the actor. The only difference is that the critic network takes both the states and the action as inputs. According to the DDPG paper [65], the actions were not included until the 2nd hidden layer of Q-network. Here, the Keras function *Merge* was used to merge the action and the hidden layer together

Figure 3.2: Critic Neural Network architecture



Source: Author

As in DDQN algorithm, a replay buffer to learn in mini-batches was also used. The replay buffer is a finite sized cache R. Transitions were sampled from the environment according to the exploration policy and the tuple (x_k, u_k, r_k, x_{k+1}) was stored in the replay buffer. Then the replay buffer was full the oldest samples were discarded. At each time step the actor and critic are updated by sampling a minibatch

uniformly from the buffer. Because DDDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transition.

Directly implementing Q-learning with neural networks was proven to be unstable in many environments. Since the network $Q(x,u|\theta^Q)$ being updated is also used in calculating the target value, the Q update is prone to divergence. The solution is a modified neural network for actor-critic and to use "soft" target updates, rather than directly copying the weights. A copy of the actor and critic networks was made, $Q'(x,u|\theta^{Q'})$ and $\pi'(x,\theta^{\pi'})$ respectively, that are used for calculating the target values. The weights of these target networks are then updated by having them slowly track the learned networks: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. This means that the target values are constrained to change slowly, greatly improving the stability of learning. This simple change moves the relatively unstable problem of learning the action-value function closer to the case of supervisioned learning, a problem for which robust solutions exist. This may slow learning, since the target network delays the propagation of value estimations.

About normalizations, they were not applied. However, when learning from low dimensional feature vector observations, the different components of the observation may have different physical units (for example, positions versus velocities) and the ranges may vary across environments. This can make it difficult for the network to learn effectively and may make it difficult to find hyper-parameters which generalize across environments with different scales of state values.

A major challenge of learning in continuous action spaces is exploration. An advantage of off policies algorithms such as DDPG is that the problem of exploration can be treated independently from the learning algorithm. There are two techniques that could be used here: parameter noise or action noise. However, is shown by [64] that add parameter noise to a neural network based actor/policy will bring better results. So, this noise will be added through dropout normalization, in which random neurons are deactivated. For each layer, the following dropout percentages were chosen. Remembering that other parameter values were not tried.

Table 2: Dropout table

Hidden Layer	Dropout
Hidden 1	5%
Hidden 2	5%

For the targets, a copy of the actor and critic networks was created and then used for calculating the target values. The weights of these target networks are then updated by having them slowly track the learned networks as shown below.

Finally, to summarize the method applied:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

4. Results

4.1 PID Controlled Response

After using a parameter grid search, the following controller constants were found to give a good response for a random step input:

$$[K_p, K_i, K_d] = [2, 0.0005, 20]$$

Remember that only the angular velocities are controlled. The x, y and z position are not controlled.

Figure 4.1: PID controller response – 3D Coordinates

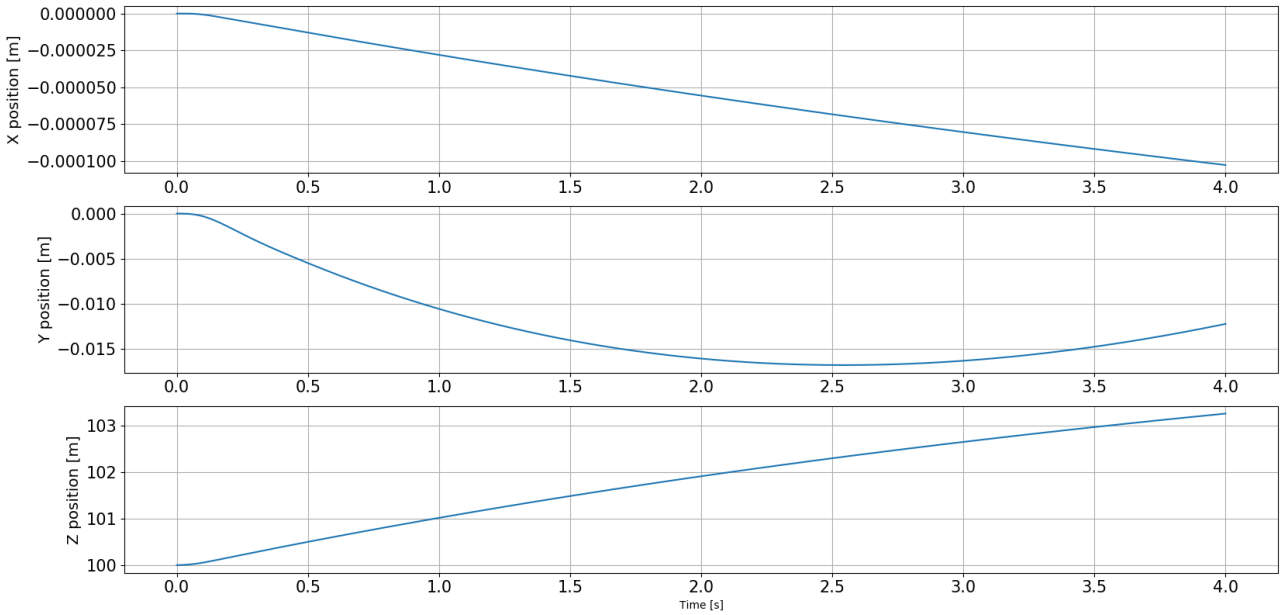


Figure 4.2: PID controller response – 3D path

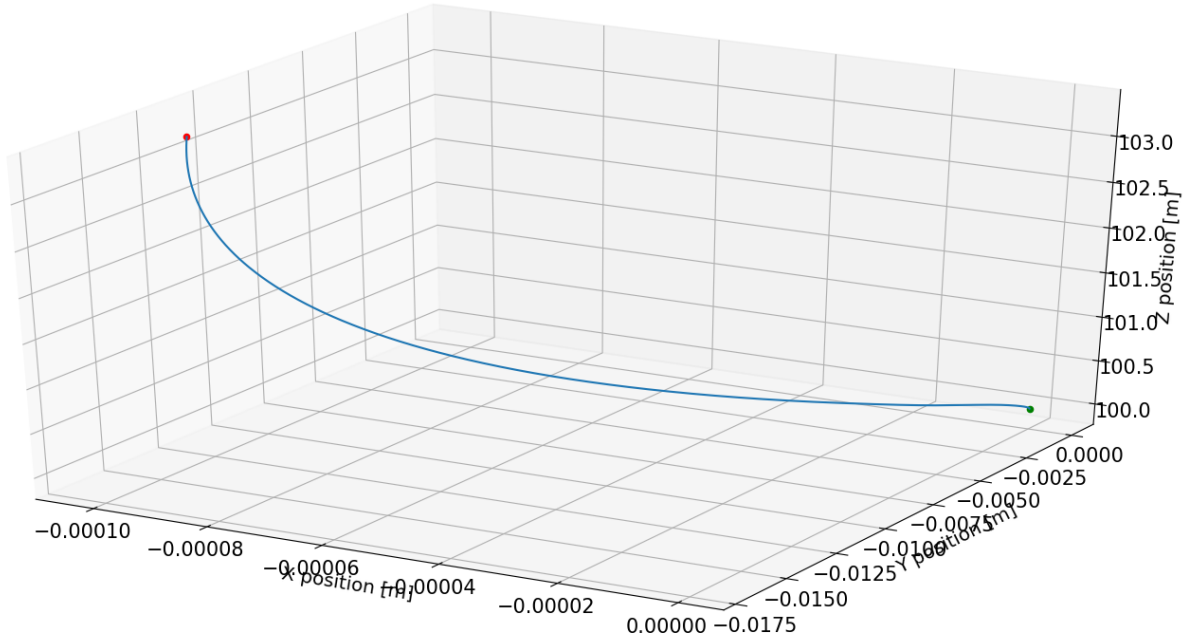
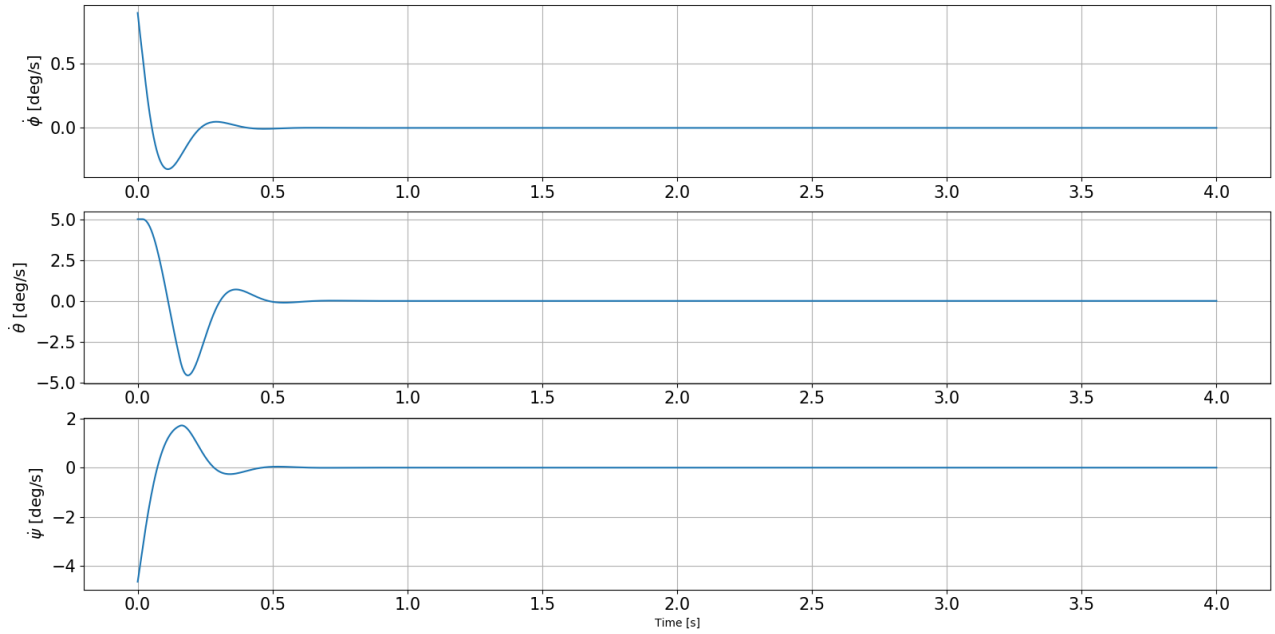


Figure 4.3: PID controller response – Angular Velocities



4.2 Reinforced Learning Response – (RL + PD controller)

Remember that only the angular velocities are controlled. The x, y and z position are not controlled.

Figure 4.4: Reinforced Learning controller response – 3D Coordinates

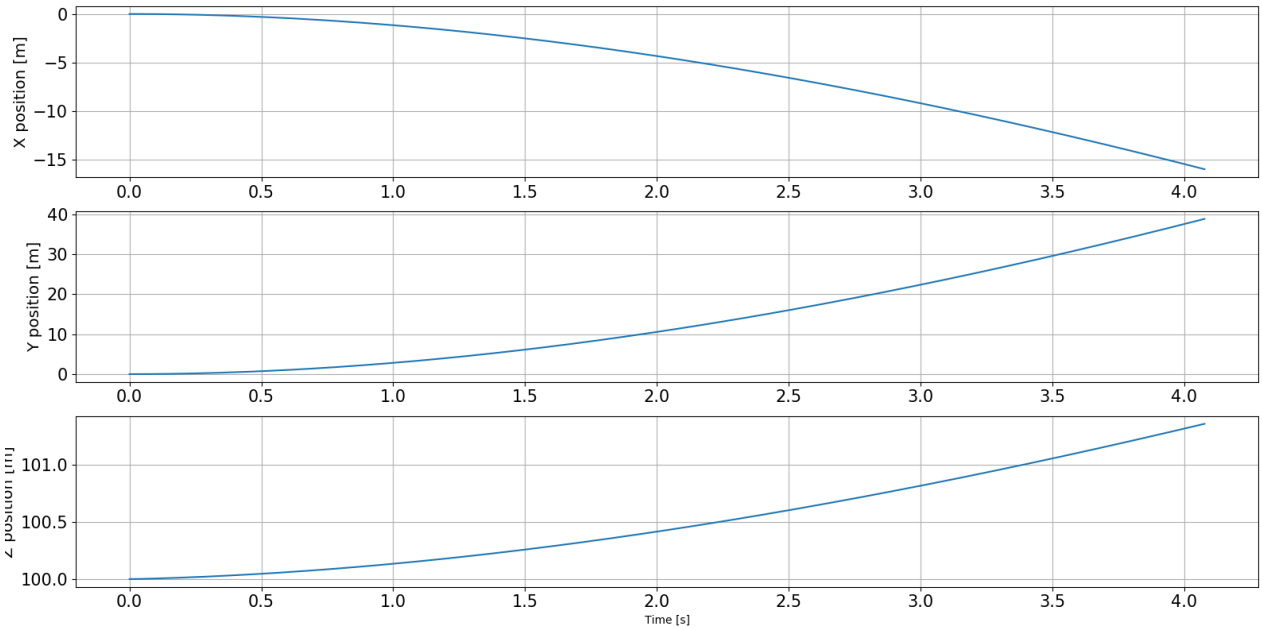


Figure 4.5: Reinforced Learing controller response – 3D path

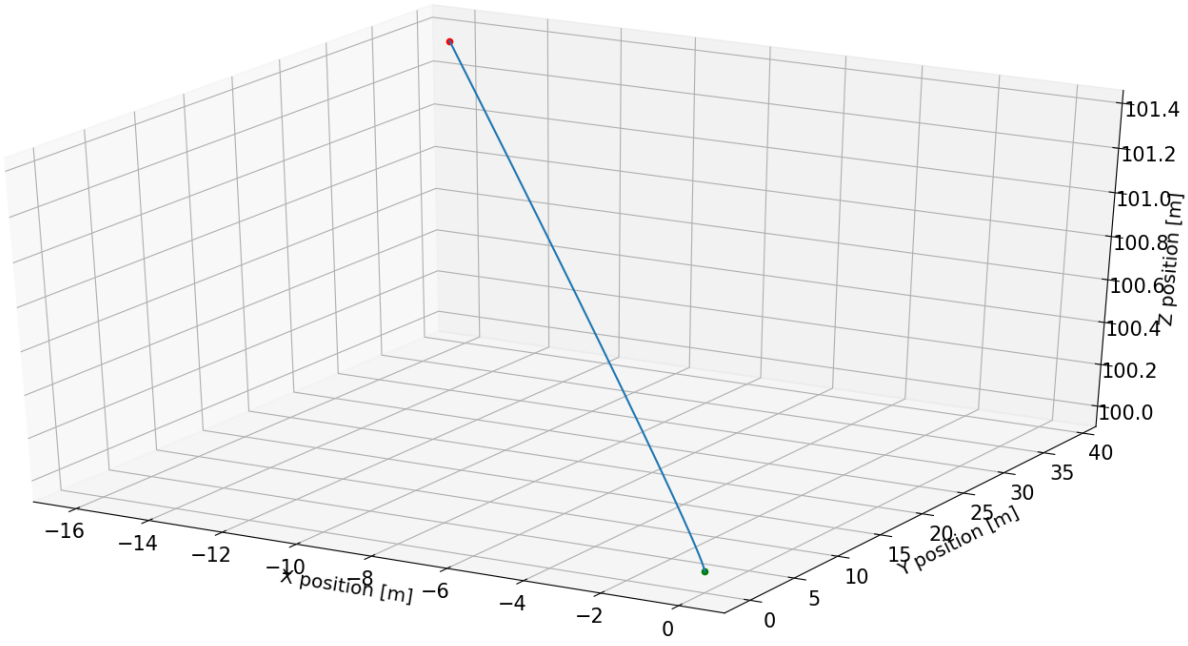
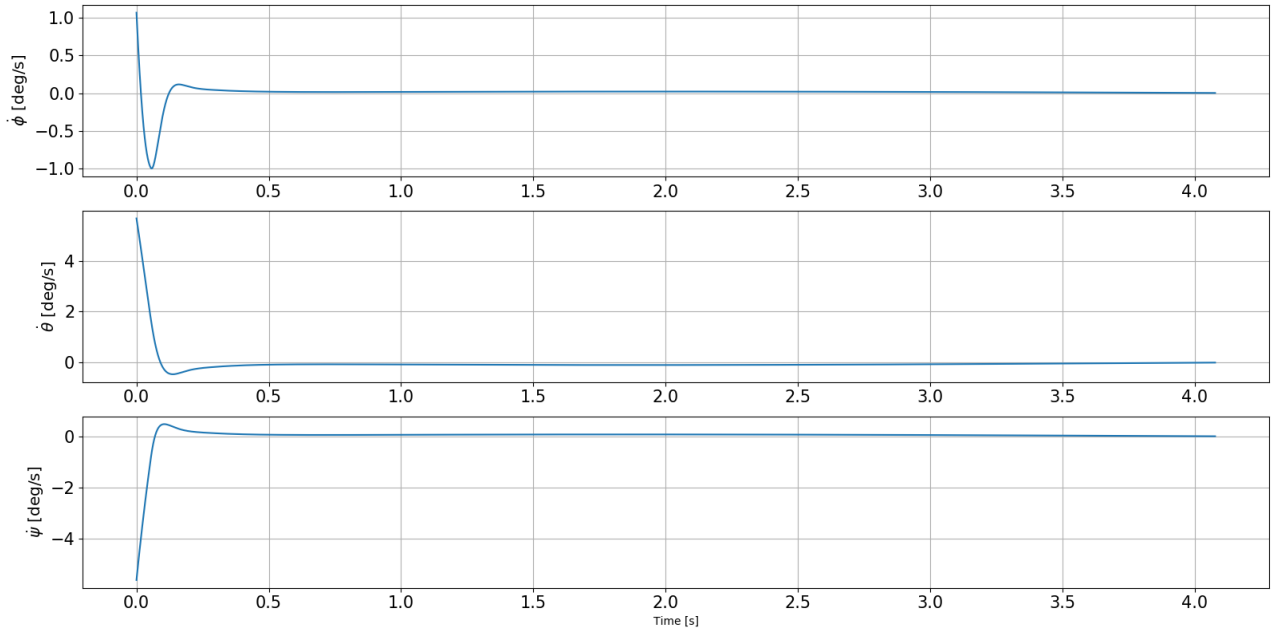


Figure 4.6: Reinforced Learning controller response – Angular Velocities



5. Conclusions

Thus, this work has shown us that is possible to control a quadcopter using Reinforced Learning techniques. However, the model only converged for small angles, what makes impossible to compare the original PID result with the Reinforced Learning results. However, we cannot say that PID is better than RL or the opposite is true because the RL stability and response to non-linear conditions, payload change as well as external perturbations were not tested.

In addition to, although deterministic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters and layer activation function type. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

For future works, is recommended to try parameter grid search optimization for the Reinforced Learning control as well as mini-batch normalization. The normalization can be helpful in several ways, according to [66]: First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

6. References

1. WHAT is a Drone? Drone vs Quadcopter. **Drone and Quadcopter**, 2019. Disponivel em: <https://droneandquadcopter.com/what-is-a-drone/>. Acesso em: 15 ago. 2019.

2. UBIRATAN, E. A origem dos vant. **Aeromagazine**, 2015. Disponivel em: <https://aeromagazine.uol.com.br/artigo/origem-dos-vant_1907.html>. Acesso em: 15 ago. 2019.

3. DOHERTY, P.; RUDOL, P. A UAV Search and Rescue Scenario with Human Body Detection and Geolocalization. **AI 2007: Advances in Artificial Intelligence** , p. 1-13, 2007.

4. DE OLIVEIRA ANDRADE, R. O Voo do Falcão. **Pesquisa Fapesp**, n. 211, set. 2013.

5. SCUSSEL, A. Pesquisa revela dados sobre o mercado de VANTs no Brasil. **MundoGEO**, abr. 2013.

6. MARTINEZ, K. The History Of Drones (Drone History Timeline From 1849 To 2019). **Drone Ethusiast**, 2018. Acesso em: 15 ago. 2019.

7. TENET, GEORGE; EX-DIRECTOR OF CIA. **Non armed surveillance missions had been taking place over Afghanistan since 2000**. [S.l.].

8. SINGER, P. W. Wired for War: The Robotics Revolution and Conflict in the 21st Century. **Penguin**, New York, p. 34, 2009.

9. WOODS, C. OK, fine. Shoot him.' Four words that heralded a decade of secret US drone killings. **The Bureau of Investigative Journalism**, p. 47-48, nov. 2012.

10. BOWDEN, M. How the Predator Drone Changed the Character of War. **Smithsonian Magazine** , nov. 2013.

11. GOLDMAN SACHS RESEARCH. **Drones**. Goldman Sachs. [S.l.].

12. Estudo Sobre a Indústria Brasileira e Europeia de Veículos Aéreos Não Tripulados. Ministério da Indústria, Comércio Exterior e Serviços - Brasil. [S.l.].

13. DIVYA, J. Exploring the latest drone technology for commercial, industrial and military drone uses. **Business Insider**, p. 13, jul. 2017.

14. CONVERTAWINGS Model A - 1956. **Aviastar**. Disponivel em: <http://www.aviastar.org/helicopters_eng/convertawings.php>. Acesso em: 10 set. 2019.

15. DARACK, E. A Brief History of Quadrotors. **Airspacemag**, maio 2017.

16. HISTORY of Quadcopters and other Multirotors. **KrossBlade Aeroespace**. Disponivel em: <https://www.krossblade.com/history-of-quadcopters-and-multirotors>. Acesso em: 12 set. 2019.

17. SILVA DE AZEVEDO, F. R. Complete System for Quadcopter Control, Porto Alegre, 2014.

18. SANTOS, M. C. P. et al. An adaptive dynamic controller for quadrotor to perform trajectory tracking tasks. **J. Intell. Robot. Syst.**, p. 5-16, 2019.

19. JAYAKRISHNAN, H. J. Position and attitude control of a quadrotor UAV using super twisting sliding mode. **IFAC Pap. Online**, p. 284-289, 2016.

20. XIONG, J. J.; ZHENG, E. H. Position and attitude tracking control for a quadrotor UAV. **ISA Trans.**, 2014.

21. NADDA, S.; SWARUP, A. Improved quadrotor altitude control design using second-order sliding mode. **J. Aerosp. Eng.**, 2017.

22. MOAWAD, N. M.; ELAWADY, W. M.; SARHAN, A. M. Adaptive PID sliding surface-based second order sliding mode controller for perturbed nonlinear systems. **In Proceedings of the 12th International Conference on Computer Engineering and Systems (ICCES)**, Cairo, Egypt, p. 19-20, dez. 2017.

23. GONZÁLEZ, I.; SALAZAR, S.; LOZANO, R. Chattering-free sliding mode altitude control for a quad-rotor aircraft: Real-time application. **J. Intell. Robot. Syst.** , p. 137-155, 2014.

24. MULIADI, J.; KUSUMOPUTRO, B. Neural network control system of UAV altitude dynamics and its comparison with the PID control system. **J. Adv. Trans.**, p. 1-18, 2018.

25. MUSTAPA, Z. et al. Altitude controller design for multi-copter UAV. **In Proceedings of the IEEE International Conference on Computer, Communication, and Control Technology.**, Langkawi, Malaysia, set. 2014.

26. SANTOS, M. F. et al. Simulation and comparison between a linear and nonlinear technique applied to altitude control in quadcopters.. **In Proceedings of the 18th International Carpathian Control Conference (ICCC).**, Sinaia, Romania, p. 234-239, maio 2017.

27. POUNDS, P. E. I.; BERSAK, D. R.; DOLLAR, A. M. Stability of small-scale UAV helicopters and quadrotors with added payload mass under PID control. **Auton. Robots**, p. 129-142, 2012.

28. SALIH, A. L. et al. Modelling and PID controller design for a quadrotor unmanned air vehicle. **In Proceedings of the IEEE International Conference on Automation,** Cluj-Napoca, Romania, maio

2010.

29. LI, J.; LI, Y. Dynamic analysis and PID control for a quadrotor. **In Proceedings of the IEEE International Conference on Mechatronics and Automation.**, Beijing, China, ago. 2011.

30. AHMED, A. H. et al. Attitude stabilization and altitude control of quadrotor. **In Proceedings of the 12th International Computer Engineering Conference (ICENCO)**, Cairo, Egypt, dez. 2016.

31. KHAN, H. S.; KADRI, M. B. Attitude and altitude control of quadrotor by discrete PID control and non-linear model Predictive control.. **In Proceedings of the International Conference on Information and Communication Technologies (ICICT)**, Karachi, Pakistan, dez. 2015.

32. BOLANDI, H. et al. Attitude control of a quadrotor with optimized PID controller. **Intell. Control Autom.**, p. 335-342, 2013.

33. THANH, H. L. N. N.; HONG, S. K. Quadcopter robust adaptive second order sliding mode control based on PID sliding surface.. **IEEE**, 2018.

34. THANH, H. L. N. N.; NGUYEN, N. P.; HONG, S. K. Simple nonlinear control of quadcopter for collision avoidance based on geometric approach in static environment.. **Int. J. Adv. Robot. Syst.** , 2018.

35. NGUYEN, N. P.; HONG, S. K. Fault-tolerant control of quadcopter UAVs using robust adaptive sliding mode approach. **Energies**, dez. 2019.

36. NGUYEN, N. P.; HONG, S. K. Position control of a hummingbird quadcopter augmented by gain scheduling.. **Int. J. Eng. Res. Technol.** , nov. 2018.

37. MILHIM, A. B.; ZHANG, Y. Gain Scheduling based PID controller for fault tolerant control of a quad-rotor UAV.. **In Proceedings of the AIAA Infotech@Aerospace**, Atlanta, USA, abr. 2010.

38. GAUTAM, D.; HA, C. Control of a quadrotor using a smart self-tuning fuzzy PID controller. **Int. J. Adv. Robot. Syst.**, out. 2013.

39. GOODARZI, F.; LEE, D.; LEE, T. Geometric nonlinear PID control of a quadrotor UAV on SE(3). **In Proceedings of the European Control Conference (ECC)**, Zürich, Switzerland, jul. 2013.

40. TAKAGI, T.; SUGENO, M. Fuzzy identification of systems and its applications to modeling and control. **IEEE Trans. Syst. Man Cybern.**, p. 116-132, 1985.

41. LIU, H.; SHI, P.; CHADLI, M. Finite-time stability and stabilisation for a class of nonlinear systems with time-varying delay. **Int. J. Syst. Sci.**, 2016.

42. ESTRADA, F. R. L. et al. LPV Model-based tracking control and robust sensor fault diagnosis for a quadrotor UAV.. **J. Intel. Robot. Syst.** , 2016, p. 163-177.

43. RANGAJEEVA, S. L. M. D.; WHIDBORNE, J. F. Linear parameter varying control of a quadrotor. **In Proceedings of the 6th International Conference on Industrial and Information Systems**, Peradeniya, Sri Lanka, ago. 2011.

44. JOUKHADAR, A.; ALCHEHABI, M.; JEJEH, A. Advanced UAVs Nonlinear Control Systems and Applications.

45. MALEKY, K. N. et al. A reliable system design for nondeterministic adaptive controllers in small uav autopilots. **Digital Avionics Systems Conference (DASC)**, p. 1-5, IEEE/AIAA 2016.

46. SANTOSO, F.; GARRATT, M. A.; ANAVATTI, S. G. State-of-the-art intelligent flight control systems in unmanned aerial vehicles. **IEEE Transactions on Automation Science and Engineering**, 2017.

47. LUND, H. H.; MIGLINO, O.; NOLFI, S. Evolving mobile robots in simulated and real environments. **Artificial life**, v. 2, p. 417-434, 1995.

48. JAGANNATHAN, S.; DIERKS, T. Output feedback control of a quadrotor uav using neural networks. **IEEE transactions on neural networks**, v. 21, p. 50-66, 2010.

49. GUIRIK, A.; BOBTSOV, A.; BUDKO, M. Hybrid parallel neuro-controller for multirotor unmanned aerial vehicle. **Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)**, 8th International Congress on. IEEE, p. 1-4, 2016.

50. SHEPHERD III, J. F.; TUMER, K. Robust neuro-control for a micro quadrotor. **in Proceedings of the 12th annual conference on Genetic and evolutionary computation**, 2010.

51. WILLIAMS-HAYES, P. S. Flight test implementation of a second generation intelligent flight control system. **infotech@ Aerospace, AIAA-2005-6995**, p. 26-29, 2005.

52. LUUKKONEN, T. Modelling and Control of Quadcopter.

53. ALDERETE, T. S. Simulator Aero-Model Implementation. **NASA Ames Research Center**, California, USA.

54. ORTEGA, M. G.; RAFFO, G. V.; RUBIO, F. R. An integral predictive/nonlinear H1 control structure for a quadrotor helicopter. **Automatica**, v. 46, 2010.

55. BOUADI, H.; TADJINE, M. Nonlinear observer design and sliding mode control of four rotors helicopter. **Proceedings of World Academy of Science, Engineering and Technology**, p. 225-230,

2007.

56. WASLANDER, S. L. et al. Quadrotor helicopter flight dynamics and control: Theory and experiment. **Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit**, ago. 2017.

57. HUANG, H. et al. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. **IEEE International Conference on Robotics and Automation**, 2009.

58. DAYANA, P.; NIV, Y. Reinforcement learning: The Good, The Bad and The Ugly, 2008.

59. GULLAPALLI, V. A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions. **Neural Networks**, v. 3, p. 671-692, 1990.

60. WILLIAMS, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. **Machine Learning**, v. 8, p. 229-256, 1992.

61. MOORE, A.; WILLIAMS, R. J. Gradient Descent for General Reinforcement Learning. **Advances in Neural Information Processing Systems 11, MIT Press**, 1999.

62. MORBIDI, F.; CANO, R.; LARA, D. Minimum-Energy Path Generation for a Quadrotor UAV. **IEEE International Conference on Robotics and Automation**, Stockholm, Sweden, 2016.

63. MANCUSO, R. et al. Reinforced Learning for UAV Attitude Control, 2018.

64. PLAPPERT, M.; HOUTHHOOFT, R.; ET. AL. Parameter Space Noise for Exploration, 2017.

65. SILVER, D.; ET. AL. Deterministic policy gradient algorithms. **ICML**, 2014.

66. IOFFE, S.; SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shif, 2015.