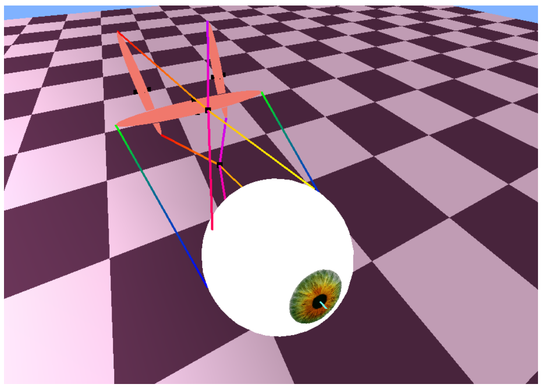


Learning Open-Loop Saccadic Control of a Biomimetic Eye using the Soft Actor-Critic Algorithm

Henrique Maria Reiche Granado

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Alexandre José Malheiro Bernardino
Dr. Adrianus Johannes Van Opstal

December 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I would like to start off by thanking my supervisors, Prof. Alexandre Bernardino, from Insituto SUperior Técnico, and Prof. John Van Opstal, from Radboud University, for their support, feedback and insight throughout the development of these thesis, specially during its final stages. To the ORIENT team, specially, Akhil John and Reza Javanmard, who gave me the push and help needed to get me started on this work. To people and things that have always been there for me... And to my family, for all the support that gave me throughout my life, including my sister, who I promised I'd include the following statement: ”Finally a big thank you to my little sister, who supported me throughout this entire process.”

Resumo

A aplicação de algoritmos de aprendizagem por reforço em robótica tem vindo a crescer ao longo das últimas décadas, especialmente na implementação de tarefas não triviais, onde o controlo de robôs com um número elevado de graus de liberdade, é bastante complicado através da utilização de técnicas clássicas de controlo. O corpo humano pode ser visto como um sistema composto de vários subsistemas complexos, e o ORIENT project pretende estudar e aprender o controlo de uma versão robótica de um destes subsistemas – o olho humano. Usando o algoritmo de soft actor-critic, este trabalho tem como objetivo ligar aprendizagem por reforço a este problema de controlo, e criar um *framework* que irá aprender o controlo em malha aberta do movimento do olho para qualquer modelo. A base do controlo implementado é inspirada no sinal de controlo do olho humano, onde um sinal de pulso é gerado, integrado e enviado através do sistema nervoso para os músculos necessários para efetuar os movimentos desejados (sacadas). A métrica para avaliar as sacadas produzidas vai ser também inspirada no sistema humano, e será usada para guiar o algoritmo de aprendizagem a resultados desejados. Esta metodologia foi aplicada a uma versão bastante simplificada do olho humano, devido a limitações de tempo, e conclusivamente o algoritmo conseguiu aprender o controlo ótimo de sacadas, para três funções diferentes de funções de pulso: as trajetórias obtidas, juntamente com as suas propriedades não-lineares assemelham-se àquelas registadas em humanos.

Palavras-chave: Soft Actor-Critic, Controlo em Malha Aberta, Olho Biomimético, Sacadas

Abstract

The application of reinforcement learning algorithms to robotics has been increasing over the last decades, specially in the implementation of non-trivial tasks where the control of robots with a high number of degrees of freedom is very difficult using classic control techniques. The human body can be seen a system composed of very complex subsystems, and the ORIENT project aims to study and learn the control of a robotic version of one of these subsystems – the human eye. Using the soft actor-critic algorithm, this work aims to link reinforcement learning to this control problem, and create a framework that will learn the open-loop control of the movement of the eye. The base for the type of control implemented is inspired on the human eye control signal, where a pulse signal is generated, integrated and sent through the nervous system to the muscles required to perform the wanted movements (saccades). The metric that evaluates the saccades produced is also inspired by the human system and is used to lead the learning algorithm to the desired results. This methodology was applied on a very simplified version of the human eye as a proof of concept. The algorithm managed to learn the optimal saccadic control strategy, for three different versions of pulse functions. The trajectories obtained, have non-linear properties similar to the ones registered in humans.

Keywords: Soft Actor-Critic, Open-Loop Control, Biomimetic Eye, Saccades

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Over the past decades, the speed of technological developments has increased tremendously. Part of these developments aim at alleviating everyone's daily burdens in life, for example, through designing machines, including robotic systems, which may eventually automate our daily routines: machines that wash our clothes, washing dishes or do vacuum cleaning; ICT gadgets that let you know whether the fridge is running out of supplies, or that can turn on the lights by a simple voice command; but also vehicles that are able to transport us around safely in cities and on highways without the need of a human driver - automation is everywhere. A major challenge for many of these technologies is that they have to reliably and safely perform under highly uncertain and noisy conditions, and in a-priori unknown environments.

One of the major conceptual advances that is guiding these developments is the statistical technique called ”machine learning”, which is defined ”as the study of statistical computer algorithms that improve automatically through experience” [1], and which manage to perform tasks that would otherwise be extremely difficult, if not impossible, to be carried out by humans. These algorithms are in the forefront of development, and their many application are used in critical professions such as medicine, where image recognition is used to diagnose patients, or artificial intelligence systems that would suggest concrete treatment for a given patient on the basis of a large amount of data [2].
At the same time, these developments run the risk to cause distress to people who fear to lose control over their lives, or lose human connection. It is with this in mind that the development of humanoid robots may be thought to better connect this complex technology with humans. Humanoid robots could be employed to perform tasks in ways that resemble humans. However, the human body is a highly complex system that is shaped by an evolutionary process during many millions of years. Mimicking such systems in a humanoid robotic system with model-based control techniques is not a trivial task.

One example of a complex subsystem of the human body is the eye. Within the ORIENT project (a EU Horizon 2020 ERC advanced grant) our group learns and studies how to control a humanoid robotic version of the human eye through the application of different control techniques [3–7]. So far, the group

has worked on different robotic models, and used classical control techniques to generate the desired movements of the robotic eye
(see, e.g. Orient's web page, at http://www.mbfys.ru.nl/∼johnvo/Orient.html).

1.2 Objectives

This thesis follows up on previous work done under the ORIENT project, and aims to develop a framework with a new methodology to study how the eye moves. Unlike the other works, this will follow a machine-learning approach for the control of the eye, where the algorithm will be put in situations where a certain movement is desired and by trial and error the algorithm will learn how to perform desired motions. Due to the exhaustive and time consuming learning process of machine learning algorithms, the model of the eye used for learning won't be as complex as the biomimetic models that were developed so far for the project, but it will be constrained to a much simpler one-dimensional model of the eye. This model will have linear characteristics, unlike the other ones, but nonetheless the methodology that will be developed will take this feature into account, and be implemented in a transferable way that can be readily adapted to the application of non-linear models. The technique can also be expanded to the problem with more degrees of freedom. It will be formulated to execute movements which will serve as a base for the algorithm to learn and improve, along with a way to induce the learning to arbitrary pre-desired solutions in order to validate the performance of the algorithm.

1.3 Thesis Outline

Chapter one gives an introduction and provides the context to the overall theme of this thesis. Chapter two will provide the background and explanation of the necessary key theory to understand the relevant concepts and links between them applied throughout this work. Chapter three will provide some history of one of the models developed for the ORIENT project and the need for simplification and validation, which supported the realization of this work. Chapter four will elaborate on how the algorithm will connect to the model of the eye, it will give a description of the various components needed for the learning process and mention extra characteristics of the training process. Chapter five will present the results of the training and accounts for their validity. Lastly, chapter six will summarize the results, discuss the achievements and encountered problems, provides suggestions for future work.

Chapter 2

Background

2.1 Human Eye

The human eye is the sensory organ that collects electromagnetic radiation from $\lambda \sim$380 - 700 nm (visible spectrum), which is absorbed by photo-receptors, composed of rods and cones located at the back of the retina, and through electrical signals it sends this information to the brain as action potentials that travel via the optical nerve [8]. Depending on the orientation of the eye, different locations of the environment are perceived. For a fixed head the eye's orientation is governed by the six extra-ocular muscles attached to the eye, split into three agonist-antagonist pairs: Superior Rectus (SR) and Inferior Rectus (IR), Superior Oblique (SO) and Inferior Oblique (IO) and Medial Rectus (MR) and Lateral Rectus (LR) (Figure 2.1) - when one of the pairs pulls to move the eye, the other tends to relax. As a higher concentration of cones is located around the center of retina, the fovea, the resolution of the images captured by the human is much higher at the center than on its periphery, and so the brain uses these 6 muscles to control the gaze direction to obtain high-resolution visual information from the environment.

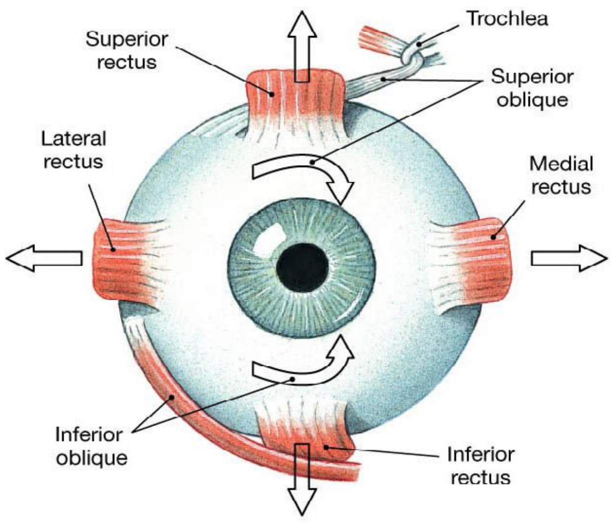


Figure 2.1: View of the 6 muscles that control the right eye

2.1.1 Saccades

The two main type of conjugate eye movements of the human oculomotor system are smooth pursuit, where the retinal slip [9] is kept to a minimum in order to track and maintain moving objects on the fovea, and saccades, which are the focus of this work. This latter eye movement type is described by a quick change of the line of sight to allow for a quick re-orientation of the fixation point. Since the brain cannot perceive clear images during fast motions [10], it's imperative to keep the duration of saccadic movements to a minimum, allowing for swift scans of the environment. Saccades with a large amplitude (30 deg or more) can reach peak velocities of 500-700 $^\circ\ s^{-1}$ [11], and most saccades have a duration of 20 to 100 ms, with inter-saccadic fixation times of 200 to 300 ms [12].

2.1.2 Neural Control

The model of the eye-plant (extra-ocular muscles, the eye ball, and surrounding tissues) has been approximated by a second-order linear over-damped system with a long time constant of about $T_1 \sim$ 200 ms, and so without any insight into the neural control, fast and precise eye movements with durations of about 40-80 ms seem at first glance counter intuitive. The fatty tissues around the eye are the major cause of this slow over damped property, far exceeding the average saccade duration [12]:

$$H_e(s) = \kappa \frac{sT_z + 1}{(sT_1 + 1)(sT_2 + 1)} \tag{2.1}$$

where $T_2 = 0.02$, $T_z = 0.07$, $T_1 = 0.2s$ and κ is a gain dependent on the three time constants. As such it can be easily seen, applying linear systems theory that a step input signal would result in an eye movement that would be far slower than a regular saccade (Figure 2.2).

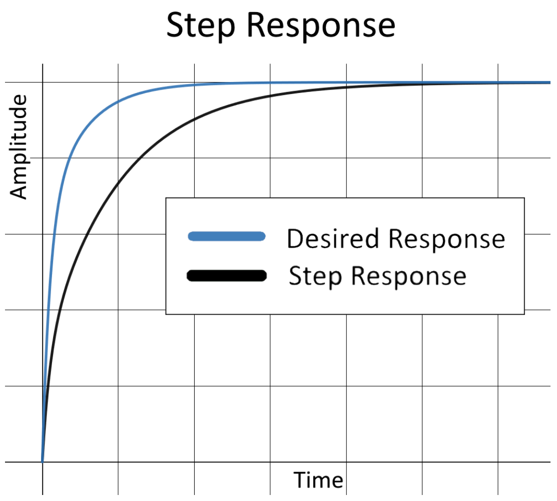


Figure 2.2: Step Response vs. Desired Response

As a consequence, the brain has to employ a non-linear pulse+step control strategy to overcome this overdamped system for fast and accurate saccades. David A. Robinson [13] proposed a model (figure

4.2) for the control of the eye which has a solid conceptual framework to study and understand the eye control system. Based on the dynamic error between the desired target and the current eye orientation, a high-frequency pulse signal is sent to the motor neurons to make the saccadic movement, and which overcomes the eye's frictional forces, and in parallel this pulse signal is neurally integrated to let the motor neurons produce the required elastic force that keeps the eye at the desired orientation. And so, the saccadic movement of the eye is caused by a pulse+step control signal, where the appropriate pulse is programmed by the brain, which is also neurally integrated:

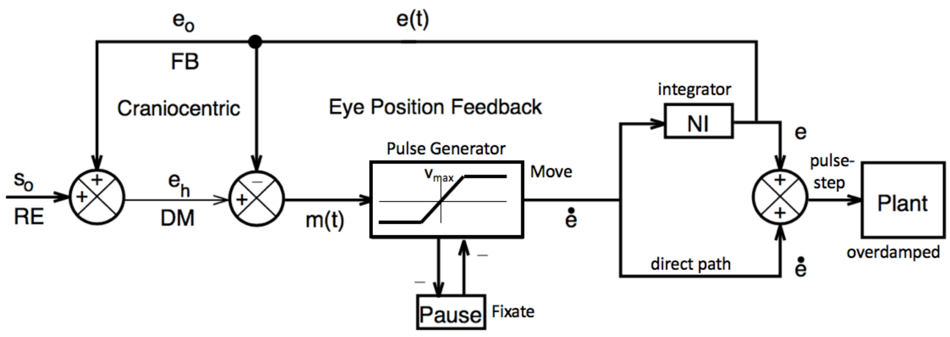


Figure 2.3: Robinson's model of the saccadic system [13]. s_0 represents the position of the target in the retina, e_h is the head-centered desired position, e_0 is the initial position of the eye, $e(t)$ is the current eye position, $m(t)$ represents the motor error and $\dot{e}$ is the pulse signal. [13]

Figure (2.4) shows empirical data from neural recordings of a single abducens motor neuron (lateral rectus eye muscle) for 11 identical saccades. The thin small lines show the individual action potentials of the neuron for each saccade, summarized below by a peri-saccadic time histogram (PSTH). It can be observed that there is a clear high-intensity pulse, followed by a slide (time constant T_z), and a step component of the signal sent to plant, which is associated with very fast step-like saccades with high reproducibility.

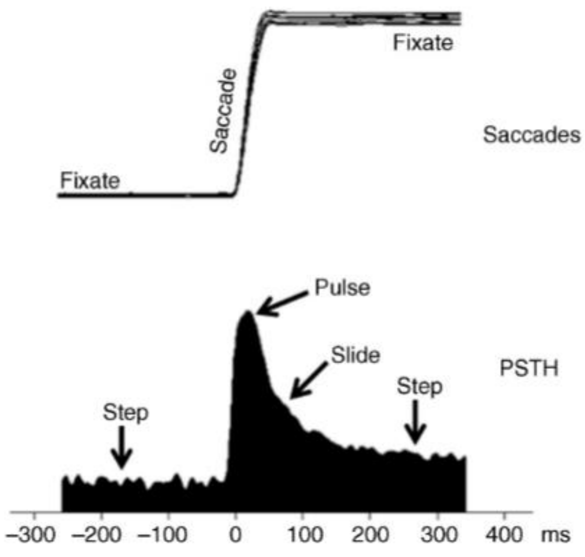


Figure 2.4: Recordings of oculormotor neuron firing for 11 identical saccades in monkeys and consequent eye trajectories [12]

2.1.3 Non-linear Dynamics

A saccade main sequence describes relationships between its properties: duration, velocity and amplitude. Skewness is another property that would describe the ratio between the time of deceleration of a saccade and its duration. These have been widely used as tool to investigate the motion of the eye, as in, designing and testing of various models of saccadic control as well as diagnosing the integrity of the saccadic system [14].

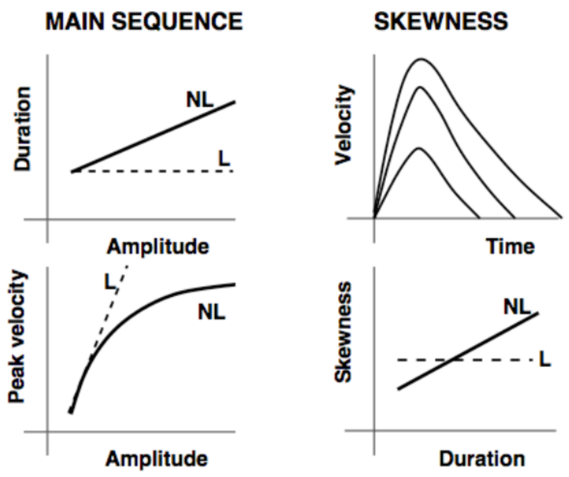


Figure 2.5: Non-linear properties of saccades

In figure 2.5, it's illustrated the very much stereotyped, across many studies, main sequence properties: the non-linear dynamic system of the saccades can be seen in all four 4 graphs. The duration of a saccade increases with amplitude, the peak velocity behaviour is linear for small saccades but as amplitude increases this value tends to saturate, and the skewness of the velocity profile increases with amplitude, this is, all saccades have approximately a constant period of acceleration (25 ms) [15], but have a larger time of deceleration as the amplitude increases, depicted in the top right graph of figure 2.5. These dynamics will be key in the work to understand and evaluate whether or not the saccadic control is adequate and if it resembles to that of the human eye.

2.2 Eye Orientation Representation

This section is introduced to give some basic understanding to the description of eye orientation that will be mentioned further into this thesis. Even though this thesis will focus mainly on horizontal angular movement, which is straightforward, expanding this concept to three dimensions will give insight into the expressions mentioned in chapters 3 and 4 which will allow to extend this work to the problem with more degrees of freedom.

2.2.1 Rotation Matrix

Any rotation can be represented by a matrix $R \in \mathbb{R}^{3\times3}$in a three dimensional euclidean space which obeys the following rules:

$$\det(R) = 1 \tag{2.2}$$

$$R^T = R^{-1} \tag{2.3}$$

And to describe a rotation around each of the three cardinal head-fixed axes, x (frontal axis; cyclotorsional rotation), y (horizontal axis; vertical rotation) and z (vertical axis; horizontal rotation):

$$R_x(\psi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{pmatrix}, R_y(\phi) = \begin{pmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{pmatrix}, R_z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

where the direction of rotation obeys the right hand rule for each axis. So to apply a rotation R to a vector v, the output vector v' could be expressed as:

$$v' = Rv \tag{2.5}$$

If there is a need to perform multiple rotations around any of these axes, a sequential multiplication of the desired rotation matrices suffices. It is important, though, to note that these sequence of rotations is not commutative, as in, performing a rotation of a vector around the x-axis, then a rotation around the y-axis and then a rotation around the z-axis could not yield the same result as performing a rotation first around the z-axis, then the y-axis and finally x-axis, or any other sequential combination.

2.2.2 Quaternion

Quaternions are an alternative way to represent rotations and 3D body orientations. They may be considered as an extension of the complex numbers, which results in a direct connection to 3D rotations with more convenient algebraic properties that 3D rotation matrices. A quaternion, q, can be represented by:

$$q = q_0 + q_1 \cdot \mathbf{I} = q_0 + q_x i + q_y j + q_z k \tag{2.6}$$

where q_0 is a scalar, $q_1 \equiv (q_x, q_y, q_z)$ is a vector of real numbers and $\mathbf{I}$ is a complex vector of i, j and k which are the fundamental quaternion units, representing the axis of rotation of the quaternion, with the following relations:

$$i^2 = j^2 = k^2 = ijk = -1 \quad \text{and } ij = k \;\; jk = i \;\; ik = -j \tag{2.7}$$

When describing rotations, it is usually done in terms of an angle ψ and a unit rotation axis $n = (n_x, n_y, n_z)$:

$$q = \|q\| \left(\cos(\frac{\psi}{2}) + n \cdot \mathbf{I} \sin(\frac{\psi}{2})\right) = \|q\| \left(\cos(\frac{\psi}{2}) + (n_x i + n_y j + n_z j) \sin(\frac{\psi}{2})\right) \tag{2.8}$$

where $\|q\| \equiv \sqrt{\sum_{i=1}^{4} q_i^2}$ is the norm (or length) of the quaternion. Without loss of generality, rotations are represented by quaternions with norm $\|q\| = 1$.

To represent a combined rotation by a quaternion $q = q_0 + q_1\mathbf{I}$ followed by another quaternion $p = p_0 + p_1\mathbf{I}$, the noncommutatice multiplicative operation between the quaternions holds:

$$pq = p_0q_0 - p_1 \cdot q_1 + p_0q_1 + q_0p_1 + p_1 \times q_1 \tag{2.9}$$

where $\cdot$ represents the dot product and $\times$ represents the cross product.
Other properties of the unit quaternions include:

$$q^{-1} = q^* \tag{2.10}$$

where the q^* represents the complex conjugate of q ($q* = q_0 - q_1\mathbf{I}$).
It is also noteworthy that the expression that represents the time derivative of a quaternion ($\dot{q}$) relates to the angular velocity, ω, of the rotating body by:

$$\dot{q} = \frac{1}{2}\omega q \tag{2.11}$$

where $\omega = 0 + \omega_1\mathbf{I}$ and ω_1 would represent the angular velocity for each of the complex axis [16].
Finally it is possible to convert the four components of a unit quaternion to nine elements of the rotation matrix, R by [17]:

$$R = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_0q_z) & 2(q_xq_z + q_0q_y) \\ 2(q_xq_y + q_0q_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_0q_x) \\ 2(q_xq_z - q_0q_y) & 2(q_yq_z + q_0q_x) & 1-2(q_x^2+q_y^2) \end{bmatrix} \tag{2.12}$$

2.3 Reinforcement Learning Framework

Reinforcement learning is a field of artificial intelligence that aims to solve problems through a system mechanism of punishment and reward. Generally speaking there will be an entity that is aimed to train, the agent, that interacts with an environment; the agent retrieves a state from the environment and applies an action to it, then after a state update it receives a reward. The agent knows nothing about the environment but has the goal of maximizing the reward obtained. Depending on the application there are various ways that the agent can make decisions. The agent could use a value-based algorithm, where the agent tries to predict the rewards obtained from applying an action to a state - from a set of actions the agent would apply the one that outputs the highest predicted reward. The agent could use a model-based approach, where it creates a simulation of the environment, predicting the follow up state from applying an action to the environment - the action applied would be obtained from simulated trajectories. Lastly, the agent could adopt a policy based learning methodology where it immediately maps a state to an action - it generates a sequence of actions that lead the agent to the final objective. The algorithm that is going to be used in this project, the soft actor-critic, will have both a policy approach and a value approach; as it does not aim to know anything about the environment this will be a model-free based approach. Before detailing the structure of the algorithm, there is the need to introduce some key concepts that will help give insight to how the algorithm works.

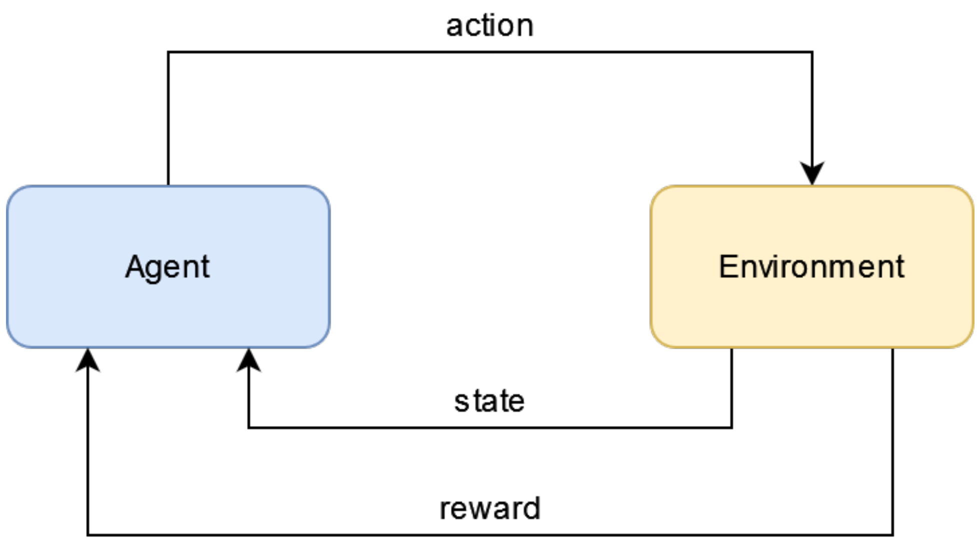


Figure 2.6: Diagram portraying the general idea behind reinforcement learning algorithms

2.3.1 Markov Decision Process

A Markov decision process is the mathematical framework that will serve as a basis for the learning algorithm. This is going to be defined as a tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ and $\mathcal{A}$ correspond to a continuous state space and continuous action space respectively, p is the probability density of the next state $s_{t+1} \in \mathcal{S}$ given the current state $s_t \in \mathcal{S}$ and the current action $a_t \in \mathcal{A}$, and r is the reward issued by the environment on each transition. Using this notation it is now possible to define the reinforcement learning problem: at each time step, $t = 0, 1, ...$, the agent will receive a representation of the state of the environment $s_t \in \mathcal{S}$ and apply an action $a_t \in \mathcal{A}$. The environment will then provide a reward r from taking the action a_t at a state s_t, $r(s_t, a_t)$. As the objective, the agent wants to maximize the cumulative reward from taking each action at each state. Then it will have to learn a policy π^* that maximizes the sum of expected rewards:

$$\pi^* = \arg\max_{\pi} \sum_{t} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t)] \tag{2.13}$$

where $\pi(a_t|s_t)$ is a probability distribution that represents the probability of taking the action a_t at a state s_t, and $\rho_\pi(s_t, a_t)$ denotes the state-action marginals of the trajectory distribution induced by a policy $\pi(a_t|s_t)$:

$$\rho_\pi(s_t, a_t) = p(s_0) \prod_{t=0} \pi(a_t|s_t)p(s_{t+1}|s_t, a_t) \tag{2.14}$$

or in other words, the probability of (s_t, a_t) independently of the succession of states and actions (trajectory) taken to get there [18]. Equation (2.13) defines the standard objective of reinforcement learning.

2.3.2 Value function and Q-function

The value function is something that is used on the value based approach; it will evaluate the benefit of being in a certain state given a policy π [18]. Since the objective is to maximize the cumulative sum of the expected reward, there is a need to represent this:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \tag{2.15}$$

where r_{t+1} represents the reward obtained at time $t+1$ and γ represents a discount factor which is in the expression to account for future uncertainty; this is some value between 0 and 1. A value of 0 would mean that there is only interest in the immediate value of the reward, and a value of 1 would be that all future would have the same importance. As such G_t is the total discounted reward for time-step t, and so as the value function of a state s_t through the use of a policy π, $V_\pi(s_t)$ is the expected sum of future rewards for state s_t:

$$V_\pi(s_t) = \mathbb{E}_\pi[G_t|s_t] = \mathbb{E}_\pi[\sum_{j=0}^{\infty} \gamma^j r_{t+j+1}|s_t] \tag{2.16}$$

Similar to the value function, the Q-function, $Q_\pi(s_t, a_t)$, is defined as the expected reward obtained from applying the action a_t to a state s_t, through a policy π:

$$Q_\pi(s,a) = E_\pi[G_t|s_t, a_t] = \mathbb{E}_\pi[\sum_{j=0}^{\infty} \gamma^j r_{t+j+1}|s_t, a_t] \tag{2.17}$$

2.3.3 Bellman Equation

The Bellman equation is a key element in reinforcement. Its use simplifies the computation of the value function, by breaking down the problem into simpler recursive sub-problems [19]. As such, the expressions for Value function can be rewritten as:

$$V_\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}|s_t] = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1})|s_t] = r(\cdot|s_t) + \gamma \sum_{s_{t+1}} \rho_\pi(s_{t+1}|s_t) V_\pi(s_{t+1}) \tag{2.18}$$

As the reward of r_{t+1} is simply the reward at state s_t, this is expressed as $r(\cdot|s_t)$, and the sum is just the definition of expected value, where $\rho_\pi(s_{t+1}|s_t)$ is the probability of jumping to a state s_{t+1} from a state s_t through a policy π.
Similarly the Q-function can be rewritten:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}|s_t, a_t] = \mathbb{E}_\pi[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})|s_t, a_t] \tag{2.19}$$

As the sum of the probabilities of taking all the possible actions is 1, $\sum_{a_t} \pi(a_t|s_t) = 1$, the value function can be rewritten in terms of the Q function and the policy π:

$$V_\pi(s_t) = \sum_{a_t} \pi(a_t|s_t) Q_\pi(s_t, a_t) \tag{2.20}$$

Similarly, it is possible to rewrite the Q-function in terms of the value function:

$$Q_\pi(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1}} \rho_\pi(s_{t+1}|s_t, a_t) V_\pi(s_{t+1}) \tag{2.21}$$

By substituting equation 2.20 into 2.21:

$$Q_\pi(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \sum_{a_t} \pi(a_{t+1}|s_{t+1}) Q_\pi(s_{t+1}, a_{t+1}) \tag{2.22}$$

And vice versa:

$$V_\pi(s_t) = \sum_{a_t} \pi(a_t|s_t) \left(r(s_t, a_t) + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) V_\pi(s_{t+1}) \right) \tag{2.23}$$

And so there is now two recursive expressions to obtain the value for the Q-function and value function.

2.3.4 Entropy

Entropy is defined as ‘lack order or predictability’ [20], and this idea will be present in the soft actor critic algorithm. For example, winning the coin game of heads or tails is much less predictable than winning the lottery. Since the probability of the first one is 50% the outcome is uncertain, whereas on the latter case the outcome of a loss is mostly certain. So, the game of heads or tails has higher entropy than the lottery. Formally speaking the entropy $\mathcal{H}(X)$ is defined as:

$$\mathcal{H}(X) = -\int_x p_d(x) \log(p_d(x)) dx \tag{2.24}$$

where $p_d(X)$ is the probability distribution of a random variable X, and the base of the logarithm can vary from application to application; here it will be kept as the standard natural logarithm for consistency reasons. In the case of the coin game, the probability of landing heads would be 0.5 ($p_d(heads) = 0.5$) and the probability of landing tails is 0.5 ($p_d(tails) = 0.5$), therefore the entropy of the game would be $H = -(0.5\log(0.5) + 0.5\log(0.5)) = 0.69$. Assuming that the chances of winning a lottery would be 1 in 14 million [21], its entropy would be $H = 1.25 \times 10^{-6}$. When the name of an expression has the word soft anterior to it it usually means that there’s the addition of this term to its standard expression.

This concept is used in reinforcement learning to promote exploration on the agents, making their actions less predictable, which is usually useful to avoid the agents getting stuck in sub-optimal solutions. An easy example would be when an agent is found in the following scenario: it has two paths to follow, A and B; if the agent chooses path A it receive a reward of 1, if it chooses path B there is a 95% of receiving nothing and a 5% change of receiving a reward of 500. If the agent had no incentive to explore, at the beginning of the training it would most likely try both paths and expect a constant reward from path A and no reward from path B. So the agent would tend to have a bias to just pick path A. The addition of entropy would incentivize the agent to try path B every so often - leading it to conclude that there can be an higher reward on path B.

2.4 Soft Actor Critic

This section is critical to understand the algorithm that is going to be used to train the model of the eye that will be explained in chapter 4. The explanation of this algorithm will be based mainly of these two papers [22, 23], where the algorithm was firstly introduced and further developed. This section will give a brief overview of its properties and also provide the key equations necessary for its understanding and implementation.

2.4.1 Overview

The soft actor-critic algorithm is an efficient design for a continuous state and action spaces using the maximum entropy framework which has been proven to be stable and robust [24]. The addition of this framework aims to reward exploration besides just learning the best policy, and although the objective is altogether modified, a temperature parameter, α, is used to regularize this entropy factor, ultimately making it possible to recover the original objective.

All-in-all, the soft actor-critic is an algorithm that maximizes reward and exploration, making use of a replay buffer, from a off-policy formulation, to improve sample efficiency and a maximum entropy to enhance stability and promote exploration. The agent will start with a random policy that will use it to apply random actions to the environment and retrieve information about it. Once it has enough samples, it will be able to distinguish better actions from worse. Then it will improve itself in a way that it will start taking better and better actions as time goes on, until a point where it is always applying the best actions to environment.

2.4.2 Framework

Putting together the standard objective of reinforcement learning (2.13) and the maximum entropy objective [24]:

$$\pi^* = \arg\max_{\pi} \sum_{t} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{2.25}$$

where α is the temperature parameter mentioned above, which will control the stochasticity of the optimal policy. As α tends to 0, the traditional objective (2.13) is recuperated. It is important to mention that the soft actor-critic models actions for a continuous space, so the policy will output actions that have some kind of continuous distribution. On our case this action will be modelled by a gaussian distribution. So, the policy will output a mean and a standard deviation, and a random sample of these values would be the action applied to the environment.

With all this in mind, the characteristics and implementation of the algorithm can be introduced. To model the soft value function, $V_\pi(s_t) = \mathbb{E}_{a_t\sim\pi}[Q(s_t,a_t) - \alpha\log(\pi(s_t|a_t))]$, the soft Q-function ($Q(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim\rho_\pi}[V(s_{t+1})]$) and the policy, there will be used 3 neural networks. The soft Q-functions will be parameterized by θ and specified by $Q_\theta(s_t,a_t)$ and the policy networks will be parameterized by ϕ, and designated by $\pi_\phi(a_t,|s_t)$. Also, these two neural networks can be identified as the critic and the actor respectively. The value function V will be parametrized by $\overline{\theta}$.

The soft Q-function parameters can be trained by minimizing the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[\frac{1}{2}(Q_\theta(s_t,a_t) - (r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}[V_{\overline{\theta}}(s_{t+1})]))^2] \tag{2.26}$$

where $\mathcal{D}$ represents the replay pool where the values of the state (s_t), the action applied to the state (a_t), the reward obtained $r(s_t, a_t)$, and the following state (s_{t+1}) are stored, which will be used to train the algorithm. As such this function can be optimized through stochastic gradient descent:

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t,s_t)(Q_\theta(a_t,s_t) - (r(s_t,a_t) + \gamma(Q_{\overline{\theta}}(s_{t+1},a_{t+1}) - \alpha\log(\pi_\phi(a_{t+1}|s_{t+1}))))) \tag{2.27}$$

where $\hat{\nabla}$ is the representation of the estimated gradient obtained.
For the update of the parameters of the value function, $\overline{\theta}$, it is done through an exponentiated moving average of the parameter θ.
The policy parameters are learned through the minimization of the Kullback-Leibler divergence, an operator that measures the difference of two probability distributions [25, 26]; the new policy will be updated towards the exponential of the new soft Q-function since this will guarantee an improvement in the soft policy in terms of the soft value [23]. Leading to a soft Bellman residual of π of:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}}[\mathbb{E}_{a_t\sim\pi_\phi}[\alpha\log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t,a_t)]] \tag{2.28}$$

To update the parameters ϕ a trick is used, which results in a lower variance estimator. The value of the policy neural network is reparameterized using the transformation $a_t = f_\phi(\epsilon_t; s_t)$ where ϵ_t is an input noise vector from some fixed distribution. From here equation 2.28 can be rewritten as:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}[\alpha\log(\pi_\phi(f_\phi(\epsilon_t;s_t)|s_t)) - Q_\theta(s_t, f_\phi(\epsilon_t;s_t))] \tag{2.29}$$

Since π_ϕ is defined implicitly in terms of f_ϕ, the gradient can approximated to:

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi\alpha\log(\pi_\phi(a_t|s_t)) + (\nabla_{a_t}\alpha\log(\pi_\phi(a_t|s_t)) - \nabla_{a_t}Q_\theta(s_t,a_t))\nabla_\phi f_\phi(\epsilon_t;s_t) \tag{2.30}$$

Lastly the term that needs to be trained is α. Again this parameter affects how much the algorithm will explore, and for some tasks it is better to explore than others; not only that, even within tasks, it is better to explore in certain situations than others - so keeping this value fixed can be ill-advised and non-trivial task to regulate manually.Instead, this parameter will be automatically adjusted by the algorithm, and the problem is formulated such that the average entropy of the policy is constrained. The residual for the update of this parameter is derived in [23]:

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_t}[-\alpha\log(\pi_t(a_t|s_t)) - \alpha\overline{\mathcal{H}}] \tag{2.31}$$

where π_t is the policy applied at time t, but this result was derived to obtain higher value in regions where the optimal policy is uncertain, promoting exploration, and lower value when the algorithm is able to make distinction between good and bad solutions.
With all these pieces is now possible to present the algorithm:

Algorithm 1 Soft Actor-Critic

```
Input: $\theta$, $\phi$                                              ▷ Initial parameters
    $\overline{\theta} \leftarrow \theta$                              ▷ Initialize target networks weights
    $\mathcal{D} \leftarrow \emptyset$                                ▷ Initialize an empty replay pool
    for each iteration do
        for each environment step do
            $a_t \sim \pi_\phi(a_t|s_t)$                              ▷ Sample action form the policy
            $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$                        ▷ Sample transition from the environment
            $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, r(s_t, a_t), s_{t+1})$   ▷ Store the transition in the replay pool
        end for
        for each gradient step do
            $\theta \leftarrow \theta - \lambda_Q \hat{\nabla}_\theta J_Q(\theta)$        ▷ Update the Q-function parameters
            $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$            ▷ Update policy weights
            $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$            ▷ Adjust Temperature
            $\overline{\theta} \leftarrow \iota\theta + (1-\iota)\overline{\theta}$      ▷ Update target network weights
        end for
    end for
Output: $\theta$, $\phi$                                             ▷ Optimized parameters
```

Above there is a simplified implementation of the algorithm, where ι represents the coefficient of discount of the exponential moving average and the λ_Q, λ_π and λ represent the learning rates or step sizes when updating the parameters θ, ϕ and α through gradient optimization methods. As it can be seen the algorithm alternates between collecting samples from the environment and updating the function approximators through sampled batches stored in the replay pool.

2.5 State of the Art

There have been many reinforcement learning algorithms created and developed, each with their own advantages and disadvantages. On one hand, there are on-policy deep reinforcement learning algorithms such as TRPO [27], PPO [28] or A3C [29], albeit stable, they have the flaw of having a poor sampling efficiency, they require new samples for each time the algorithms try to learn/ improve, which becomes extremely as the number of step required to learn the optimal policy increases. On the other hand there are off-policy algorithms, such as DDPG [30] and its improved version TD3 [31], which make use of a replay buffer to make them algorithms with good sample efficiency. However these are very brittle when it comes to hyperparameter tuning: lousy values for learning rates, exploration constants

and other properties are enough to make algorithms unstable and unable to converge. And so new reinforcement learning algoirthms have been emerging that work well when applied to real life robotic tasks. Moreover the implementation of these techniques to robots with high complexity, numerous degrees of freedom and which perform tasks considered more than intricate, are highly effective and somewhat straight forward to implement, making traditional control techniques obsolete. In [32], the algorithm Policy Learning by weighting exploration with the returns (PoWER) is showcased and is effective in performing tasks that require an initial demonstration of the policy needed to perform - the robot is physically moved to perform successfully a task - the algorithm then learns by reproducing and adapting from the initial demonstration. In tasks such as the tether-ball target hitting in simulation, the under-actuated swing-up and ball in a cup managed to have successful outcomes. In [33], it is also successfully showcased the use of 3 expectation-maximization reinforcement learning algorithms to perform 3 different tasks in real life robots: pancake flipping, minimization of energy on a bipedal robot and archery.
Regarding the use of the soft actor-critic algorithm to robotics, in [23], the algorithm has been used to perform two different tasks: the first one was a quadrupedal robot which learned the policy for walking on a flat surface, but when this was placed on a environment with slopes, the policy managed to generalized to such scenario and perform well. The second application was to perform a task where it was required to move a valve like object to the correct position, where the input of the algorithm was solely the feed from a camera - the position of the motors were not provided to the policy. There is also the application of this algorithm on a dual arm robot which aims to make movement avoiding possible collisions.
The application of this algorithm to vision related tasks has conceived works that only study patterns of gaze allocation: in an environment that require constant eye movement - to navigate a sidewalk with obstructions [34] - or how it moves while reading, does it read word by word, or keeps a constant movement [35].
The works that precede this one [3–7] have only focused on the control of a biomimetic eye through the application of linear model-based control techniques to the model. This method achieved good results for the control of the various eye plants, but had the disadvantages of a needed new implementation of the control system for every single model. This thesis aims to provide a framework for the control of this biomimetic eye using a free-model reinforcement learning approach, which unlike the previous approaches, should only require one implementation, but a thousands if not millions of episodes of training to obtain a control system.

Chapter 3

Eye Model

This chapter will provide the context necessary to understand the model that will be used for training: how the models that came before it are similar mechanically to the human eye, and why the simplification that was need, wouldn't invalidate the training of the soft actor critic algorithm on its predecessor models.

3.1 Design

The model that's going to be first used to apply the algorithm to learn its motor primitive parameters will be the one developed by Miguel Lucas [4], which consists of a semi-sphere attached to a ball-joint mechanism and connected through 6 elastics to aluminum plates attached to 3 different motors - 1 horizontal and 2 vertical, forming 3 pairs of agonists and antagonistic muscles. The elastics attached directly to the horizontal plates are supposed to mimic the MR and LR muscles, while the other elastics go through intermediate points and are supposed to mimic the other four muscles. The difference in the tensions between the elastics are what is going to introduce movement to the eye, as a result of applying rotations to the motors.

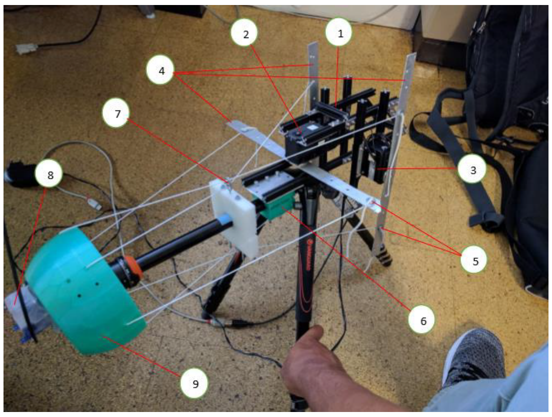


Figure 3.1: Real Biomimetic Eye Model: 1- Motor nº1 (IO SO); 2- Motor nº2 (MR LR); 3- Motor nº3 (SR IR); 4- Motor aluminium arms; 5- Two out of six washers; 6- Compatibility part (3D printed adaptor); 7- Top modified eye screw; 8- LPMS-CU accelerometer sensor; 9- 3D printed eyeball [4]

3.2 Simulator

To be able to apply the algorithm in a desirable fast and accessible environment the simulator developed by Carlos Tavares [5] was essential. He took the simple approach of building the simulator on top of Newton's Second Law applied to rotations (equation (3.1)), and assuming the elastics behave like simple stretched springs - Hooke's Law [36]:

$$T = I\alpha \tag{3.1}$$

$$F = -kx \tag{3.2}$$

where T represents a torque, I the inertial tensor of a body, and α its angular acceleration. k represents the coefficient of restitution for a spring like object, and x its displacement from a equilibrium/relaxed position. For an elastic referenced with the index i, with a unstretched length of l_{0_i} of let's call the point

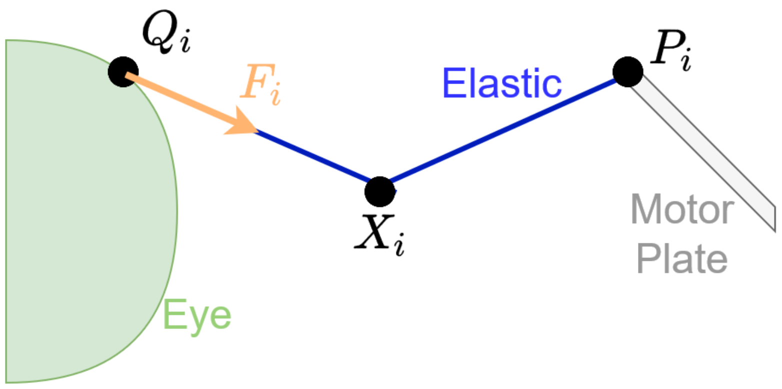


Figure 3.2: Geometric schematic of the variables involved in the computation of the forces exerted on the eye by a single elastic. P_i represents the insertion point of the elastic on the motor plate, X_i is an intermediate point that the elastic passes through and Q_i is the insertion point of the elastic on the eye.

where the elastic is attached to the aluminum plate P_i, and Q_i the point where the elastic is attached to eye. If an elastic goes through an intermediate point X_i, the force F_i applied to the eye is (Figure 3.2):

$$F_i = -k(|P_i - X_i| + |X_i - Q_i| - l_{0_i})\hat{d}_i \tag{3.3}$$

$$\hat{d}_i = \frac{(X_i - Q_i)}{||X_i - Q_i||} \tag{3.4}$$

where $\hat{d}_i$ is the unit vector representing the direction of the force F_i. If there's no intermediate point as a simplification $Q_i = X_i$. For a reference frame centered at the center of rotation of the eye, the torque applied by all the elastics T_{ela} on the eye is:

$$T_{ela} = \sum_i Q_i \times F_i \tag{3.5}$$

where $\times$ represents the cross product. The total torque, T is not only composed by the elastics, so it's necessary the addition of its other 2 components, gravitational torque T_g, and friction torque T_f:

$$T = T_{ela} + T_g + T_f \tag{3.6}$$

$$T_f = -\beta\omega \tag{3.7}$$

$$T_g = C_m \times F_g \tag{3.8}$$

where β is a friction coefficient, ω is the angular velocity, F_g is its gravitational force, C_m is the position of the center of mass of the eye, which might have to be updated if it's off the center of rotation:

$$C_m = RC_{m_0} \tag{3.9}$$

where C_{m_0} represents the position of the center of mass in relation to the center of rotation at the start, and R is obtained through equation 2.12, using the quaternion q representing the orientation of the eye. Using now equation 3.1, and knowing the moment of inertia of the eye I it is now possible to obtain its angular acceleration. Although it is also important to take into account that the inertial tensor will change values depending on the orientation, so this needs to be updated as well. From a certain initial orientation with a value of I_0 for the tensor, the updated initial tensor I_u can be updated using the following equation:

$$I_u = R^T I_0 R \tag{3.10}$$

By integrating this value it's possible to obtain the angular velocity ω, which in quaternions would be equation 2.11. Where ω_q is the angular velocity expressed as a quaternion (with the scalar value as 0), q is the quaternion orientation of the eye and $\dot{q}$ is its quaternion angular velocity. Finally integrating this value will give the new eye orientation as a quaternion. Below there is a simplified algorithm on how all of these equations come together to update the orientation of the eye:

Algorithm 2 Orientation Update

```
Initialize parameter vectors:
$\tau$                                              ▷ Motor Position
$Q$                                                 ▷ Insertion point on eye
$P$                                                 ▷ Insertion point on motors
$X$                                                 ▷ Intermediate points
$C_{m_0}$                                           ▷ Center of Mass
$I$                                                 ▷ Moment of Inertia
$q$                                                 ▷ Eye orientation in quaternions
$\omega$                                            ▷ Eye angular velocity in $rads\ s^{-1}$
while Simulation is running do
    $\tau \leftarrow Update\_Motor\_Position()$          ▷ Updates the Motor Position from external inputs
    $P \leftarrow Update\_Motor\_Insertion\_Points(\tau)$      ▷ Updates position of insertion points on motor
    $Q \leftarrow Update\_Eye\_Insertion\_Points(q, Q_0)$      ▷ Updates position of insertion points on eye
    $T_{ela} \leftarrow Compute\_Torque\_From\_Elastics(P, X, Q)$    ▷ Equation 3.5
    $T_{grav} \leftarrow Compute\_Gravitational\_Torque(C_{m_0}, q)$  ▷ Equation 3.7
    $T_f \leftarrow Compute\_Friction\_Torque(\omega)$              ▷ Equation 3.8
    $T \leftarrow T_{ela} + T_{grav} + T_f$                          ▷ Equation 3.6
    $\alpha \leftarrow Compute\_Angular\_Acceleration(T, I, q)$      ▷ Equation 2.12 and 3.1
    $\omega \leftarrow Compute\_Angular\_Velocity(\alpha)$             ▷ Integration
    $\dot{q} \leftarrow Compute\_Quaternion\_Angular\_Velocity(\omega, q)$  ▷ Equation 2.11
    $q \leftarrow Update\_Orientation(\dot{q})$                     ▷ Integration
end while
```

Figure 3.3 will show a visual representation of the simulator built using Pyglet [37], along with the identification of what elastics would represent which muscles on a human eye.

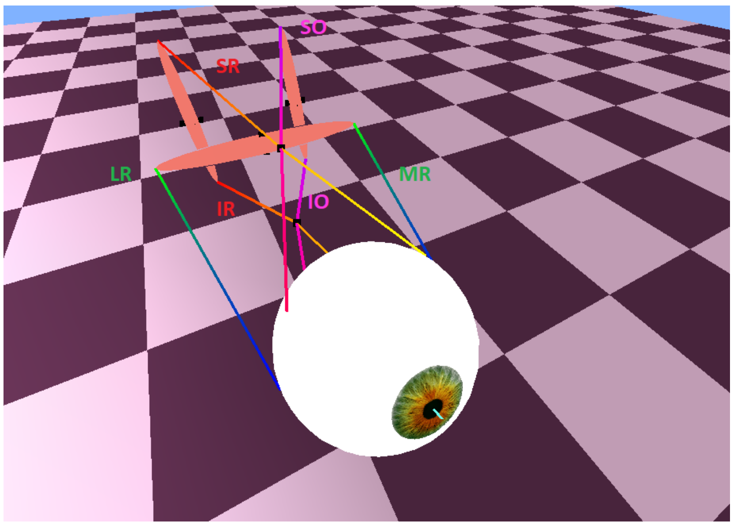


Figure 3.3: Visual representation of the simulator built using Pyglet

3.3 Model Simplification

The control the simulator model, is already a complicated task, and on top of that machine learning is very time consuming, so this model will have various simplifications that will serve as proof that reinforcement learning works. The first simplification is that only horizontal saccades will be considered - implying that the horizontal motor will be the only one used since it is uniquely responsible to changing the horizontal orientation of the eye (Figure 3.4). The next simplification has to do with the simulator model as a whole; the simulations done are discretized with a constant time step - a value which is paramount to define the precision of the simulation with a similar model in reality as well as its speed. These two factor trade-off with each other, this is, for a more accurate simulation with reality, a longer time will be taken simply because more time steps will be required. With this in mind, and taking into account that it's necessary to go through every single step of the simulation to know the behaviour of the saccade trajectory, since the model is non-linear. As reinforcement learning requires thousands if not millions of simulations to train neural networks, it will take a long time to obtain results using this physical simulator model, therefore, this model will be approximated to a first order linear model allowing for faster results using analytic methods.

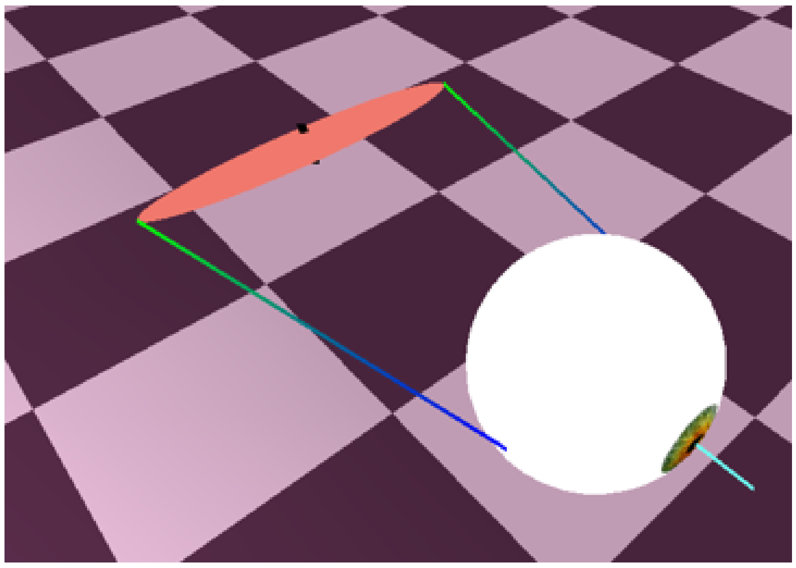


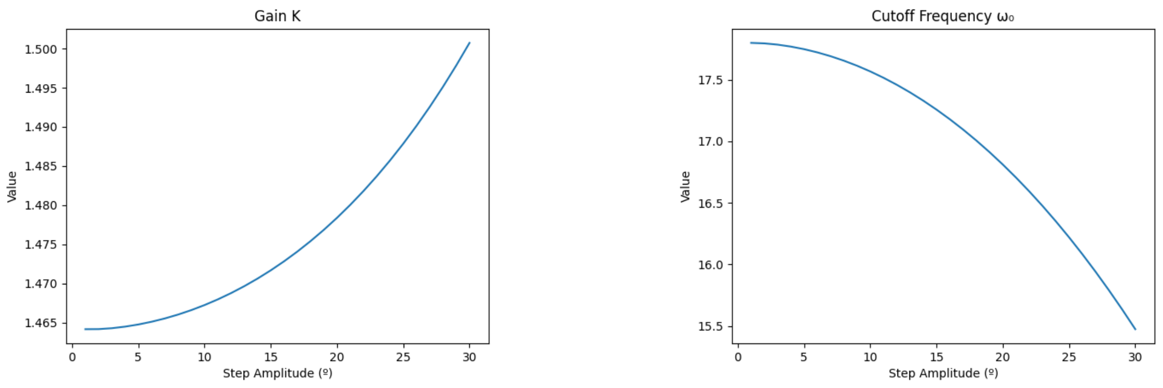
Figure 3.4: Visual representation of the simulator only for horizontal saccades

3.3.1 First Order Linear Approximation

The new model of the eye will be approached by a first order linear approximation or, first order Low-Pass Filter (LPF), with a cutoff frequency of ω_0 and a gain, K:

$$H(s) = K\frac{\omega_0}{s+\omega_0} \tag{3.11}$$

These 2 parameters can be obtained by doing a system identification of the system, applying step commands to the motor with different amplitudes and using a Least Square Estimator, it was possible to determine K and ω_0 with respect with amplitude:



(a) Value of K vs. Step Amplitude (b) Value of ω_0 vs. Step Amplitude

Figure 3.5: The values of K (left) and ω_0 (right) as a function of the step amplitude applied as inputs to the motor

In Figure 3.5, it's possible to recognize the non-linearity of the system in both graphs. If the system was linear both graphs would present horizontal lines - from the value of K, the final orientation of the eye would be proportional to the steady state position of the motor, and from the value of ω_0, the duration of the movement would be the same independently of the step amplitude. To select a value of K for the LPF it was decided to simply pick a value that would minimize the least square error from all the values for each step size, giving a value of $K = 1.4758$. For the value of ω_0 a different strategy was applied: the time step used for these simulations was of 0.001 seconds, meaning that the step of 10° would require a speed of 10000 $^\circ$ s^{-1}, which is a reasonable upper bound for the top speed of the motor, which will be essential to determine certain parameters further ahead - giving a value of $\omega_0 = 17.5$. So, from now on this will only be the only model considered for training, applied only for horizontal saccades, where changes in angle of yaw will be considered.

There are 2 concepts important to mention: the first one is the irrelevancy of the values selected for the problem, independently of what these are, the behaviour of the Soft Actor Critic Algorithm would be the same, to maximize the reward. What would change in the end would be the values to which the neural networks would converge to. Secondly, as it will be seen in the following sections, there will be a parameter that is going to be trained that would deal with the non-linearity of K, and another parameter that would deal with the non-linearity of ω_0 in the non-linear system.

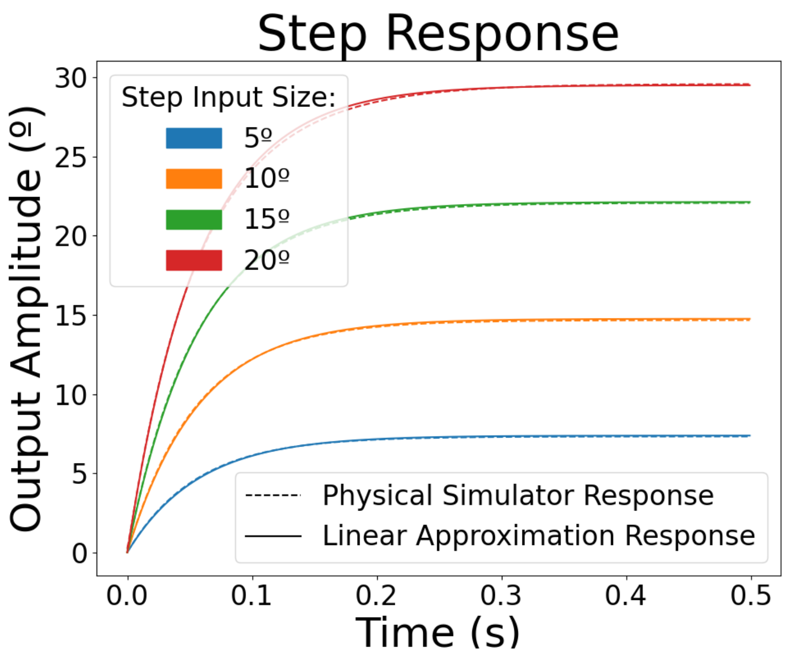


Figure 3.6: Validation of the first order linear approximation

In figure 3.6 there is the step response comparison between the physical simulator and its first order linear approximation. As it can be seen, the approximation can be considered valid since for the same sized inputs both saccades trajectories are very close to each other.

Chapter 4

Implementation

The objective of this work is to make use of the Soft Actor-Critic to learn the saccadic control of an eye model, through a process of trial and error - same way as infants do [38]. As such the problem is going to be framed in a way that resembles the human control system. When we perform saccades, the brain knows the internal orientation of the eye and predetermines its final orientation based on the position of the desired target. It then sends signals to the 6 internal ocular muscles, to extend or contract, to move the eye to the desired position. Based on the alignment between the final eye orientation and the target, further movements to the eye might be done. Even though the brain performs corrections when needed, as a simplification of the problem, the control will be done in open-loop, which actually yields similar results as the ones identified in humans [39, 40]. So in this chapter it will be described how the Soft Actor-Critic algorithm is incorporated in a reinforcement learning framework, and how it will perceive, execute and improve its saccadic control through the definition of the following key components: state, action and reward. Lastly it will be provided some insight into various parameters that had the need to be defined for the algorithm to work.

4.1 Agent

It has been established that the environment will be the first order linear approximation which will interact with the agent - the Soft Actor-Critic algorithm.

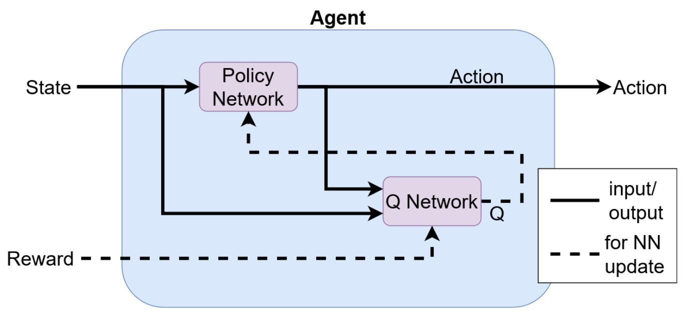


Figure 4.1: The Soft Actor-Critic Algorithm as the agent in reinforcement learning

As seen in Figure 4.1, this version of the algorithm will only require 2 neural networks (NN) because the control is done in open-loop - only one action is applied to the model, so there will be no future rewards. These 2 neural networks are the policy network which will define an action from retrieving a state, and the Q network which will evaluate the action applied to the state by outputting a Q value, which is an approximation of the reward. This Q value will be used to improve the policy network.
What is needed to do now is to define what is the state, the action and the reward.

4.2 State

Since the control will be done in open loop, the state fed into the policy network will just be the initial eye orientation, and the desired eye orientation. The state vector is represented by a tuple of quaternions parameters, (q_{i_0}, q_{i_x}, q_{i_y}, q_{i_z}, q_{d_0}, q_{d_x}, q_{d_y}, q_{d_z}), where the initial eye orientation quaternion $\mathbf{q_i} = q_{i_0} + (q_{i_x}, q_{i_y}, q_{i_z}) \cdot \mathbf{I}$, where $\mathbf{I}$ is a complex vector of i, j and k, and the final desired orientation is $\mathbf{q_d} = q_{d_0} + (q_{d_x}, q_{d_y}, q_{d_z})$. q_i is a value retrieved from the plant, and q_d is a value generated randomly from episode to episode. For contrast, if a closed-loop approach were to be implemented, for each step in time a new state would be fed into the network, since the eye moved its quaternion orientation changed - a new state would have kept the parameters for the desired orientation but changed the ones about the initial or previous orientation.

4.3 Action

The action applied to the system is what will change the orientation of the eye. In the real model, this would represent a change in the position of the motor. Since the control will be done in open-loop it is necessary for the output of the policy network to define a movement of the motor from an initial time until infinity. This could be done by approaching many positions of the motor in time as outputs of the actor network, as in, the first output neuron would define the position of the motor at time = 0 s, the second would define the position at time = 0.1 s, the third at time = 0.2 s, etc. and the position of the motor for instances of time in-between the times defined by the output neurons, would be approached by straight lines. This method unfortunately would require an unknown amount of neurons with an unknown timescale, which would exponentially complicate the size and complexity of the neural networks used.
Instead the approach taken will be to define a function $\tau(t)$ which will give the position of the motor at time t, and it will have a certain predefined overall shape which can be controlled using certain parameters. For example, the function $A \sin{(Bt)}$ will "always" have the shape of a wave, but its amplitude can be controlled by changing the parameter A, and changing B will change its frequency. So, in this case, by having only two parameters as outputs of the policy network, we would have a function defined continuously over its entire time domain. Moreover, this approach, along with the linearization done the model, will allow to have analytical solutions for its behaviour, as it will be explained further ahead.
This function $\tau(t)$ is inspired by the model of saccades established by David A. Robinson [13], and already described in section 2.1.2, such that when a saccade has to be done a pulse is generated which

with the addition of the neural integrator will output a pulse-step command that will direct the gaze to the desired position.

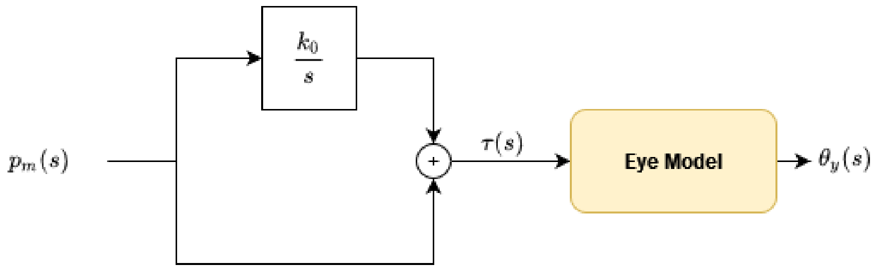


Figure 4.2: Pulse-step generator

Above is a diagram of the pulse step generator, where $p_m(t)$ is a function that defines a pulse, which goes through an integrator with gain k_0 and is added to itself to make the motor command $\tau(t)$, which is then applied to the model of the eye which will output a saccade with yaw $\theta_y(s)$. If a simple step function was defined for the motor command of a saccade, it would follow a linear behaviour on the Low Pass Filter model - saccades would take the same time, due to the modelling of the plant, independently of their amplitude. The introduction of a pulse generator and an integrator is necessary to make faster saccades than what otherwise would be possible.

$$\tau(s) = p_m(s)(1 + \frac{k_0}{s}) = p_m(s)\frac{s + k0}{s} \tag{4.1}$$

The equation 4.1 shows the relationship between the motor command τ applied and the pulse generated p_m in the Laplace domain which has the aim of producing faster movements than the ones allowed by the plant. As in, when such commands are applied to the Low-Pass Filter plant, $H(s)$, it outputs a value for the yaw θ_y:

$$\theta_y(s) = \tau(s)H(s) = (p_m(s)\frac{s + k_0}{s})(K\frac{\omega_0}{s + \omega_0}) = p_m(s)K\omega_0\frac{s + k_0}{s(s + \omega_0)} \tag{4.2}$$

As it can be seen, if $k_0 = \omega_0$, the zero that comes from the integrator and the pole that comes from the plant cancel each other. This will give a saccade with $\theta_y(s) = K\omega_0\frac{p_m(s)}{s}$ which by the definition of a pulse being a signal with an increase in amplitude from a baseline followed by a decrease in amplitude back to the baseline line value [41], will produce a step-like function based uniquely off the pulse properties - the speed of the saccade will be dependent only on the pulse generator, the plant will have no influence. On the model that will be used, it's already been defined the value of ω_0 and therefore this the value of k_0 could be also set as that value, however, one of the goals of the project is to provide a framework that would also work on a any other model, including non-linear ones, therefore k_0 will be one of the parameters of τ that will be trained by the policy network. As a preview, it is expected that on the LPF model, the value of k_0 will tend to the value ω_0 for every saccade size, and if this were trained on a non-linear model it would be expected for this value to change according to the size of the saccade - from section 3.3, figure 3.5 (b) ω_0 would decrease (it won't be verified in this work).

Now, all that's needed to do is to define the shape of the pulse: there are many ways that this could be defined, such as a rectangular pulse, Gaussian pulse, sinc pulse, etc. but to simplify the problem it will be considered a pulse with function $p_m(t) = Ae^{-Bt}f(t)$, because it is continuous, differentiable and has a simple conversion to the Laplace domain:

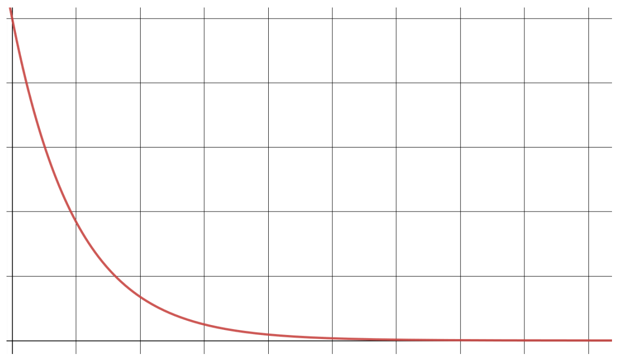


Figure 4.3: Qualitative plot of the function Ae^{-Bt}

Figure 4.3, shows the shape of the function Ae^{-Bt} which only demonstrates a decreasing behaviour and if such function were to be applied to the real model starting at a steady state, would mean that motors would instantaneously change orientation in a discrete manner at the beginning followed by a smoothed decreasing movement. The function $f(t)$ comes in to correct this unrealistic behaviour, it will provide the smooth increase from steady state to a peak amplitude. Before defining this function it is important to mention that there will be no parameters from this function that will be trained by the network, as such there will only be 3 parameters estimated by the actor network: A, B and k_0, where A would correspond to the height of the pulse a value that will be positive when positive sized saccade are produced and negative otherwise, B would represent the decay of the exponential which will be always a positive value which will produce faster saccades the higher this is. Since the goal would be to produce faster saccades than the ones limited by the plant, this value would be required to be higher than the value of the cut-off frequency of the plant ω_0. Lastly k_0 is the gain of the integrator which optimally, as said before, will be the same as ω_0. Now, f(t) could have also other parameters but at most would come/be defined from these 3 parameters, which this will be clarified below. With this in mind, the model will be trained using 3 different variants of f(t).

4.3.1 Variant 1

The first variant that will be trained is simply $f(t) = t$, so there are no extra parameters needed to be calculated:

$$p_m(t) = Ae^{-Bt}t \tag{4.3}$$

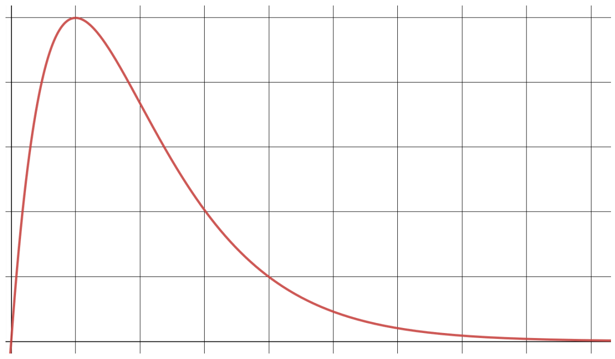


Figure 4.4: Qualitative plot of the function $Ae^{-Bt}t$

This type of pulse would give a function for the motor command $\tau(s)$:

$$\tau(s) = A\frac{s+k_0}{s(s+B)^2} \tag{4.4}$$

Which when applied to the plant:

$$\theta_y(s) = \tau(s)H(s) = (A\frac{s+k_0}{s(s+B)^2})(K\frac{\omega_0}{s+\omega_0}) = AK\omega_0\frac{s+k_0}{s(s+B)^2(s+\omega_0)} \tag{4.5}$$

Which in the time domain would give:

$$\theta_y(t) = AK\omega_0(\frac{k_0}{B^2\omega_0} + \frac{2Bk_0 - B^2 - k_0\omega_0}{B^2(B-\omega_0)^2}e^{-Bt} + \frac{k_0 - B}{-B(\omega_0 - B)}te^{-Bt} + \frac{k_0 - \omega_0}{-\omega_0(B-\omega_0)^2}e^{-\omega_0 t}) + \theta_0 \tag{4.6}$$

Where θ_0 would correspond to the initial orientation of the eye, which in steady state can be easily seen that is proportional to the initial position of the motor F, through the variable K - ($\theta_0 = KF$).
Before moving on to the next variant, it is important to note that before training it is necessary to define search boundaries for these parameters, as in the algorithm can only look for solutions within the defined ranges. Unfortunately depending on the function $f(t)$ the maximum value of A could vary substantially and so a different boundary would have to be defined for the different variants. To solve this problem a easier and more intuitive approach will be taken - transforming equation 4.4 in the Laplace domain will give:

$$\tau(t) = A(\frac{k_0}{B^2} - \frac{k_0}{B^2}e^{-Bt} + \frac{B-k_0}{B}te^{-Bt}) + F \tag{4.7}$$

Where F corresponds to the initial position of the motor. This function can be separated into its pulse p_m and step s_m components:

$$\tau(t) = p_m(t) + s_m(t) = Ate^{-Bt} + \frac{Ak_0}{B^2}(1 - e^{-Bt} - Bte^{-Bt}) + F \tag{4.8}$$

By looking at the step portion of the equation above it can be seen that the fraction $\frac{Ak_0}{B^2}$ represents the amplitude of the step and if we were to consider a final position of the motor to be D:

$$\lim_{t\to\infty}\tau(t) = p_m(t) + s_m(t) = Ate^{-Bt} + \frac{Ak_0}{B^2}(1 - e^{-Bt} - Bte^{-Bt}) + F \Rightarrow D = \frac{Ak_0}{B^2} + F \tag{4.9}$$

With this method the training of the parameter D could be done instead of the parameter A, which has clearer defined and constant bounds across all variants - these could simply be the minimum and maximum orientations of the motors, for example, in the real model if the motor was oriented over a certain threshold the elastics could snap - these limits will be set to be between $-80\ ^o$ and $80\ ^o$ which are values way above the ones required for the saccade sizes that will be trained.

4.3.2 Variant 2

This variant will be characterized by the use of smooth step:

$$f(t) = 1 - e^{-Ct} \tag{4.10}$$

$$p_m(t) = Ae^{-Bt}(1 - e^{-Ct}) \tag{4.11}$$

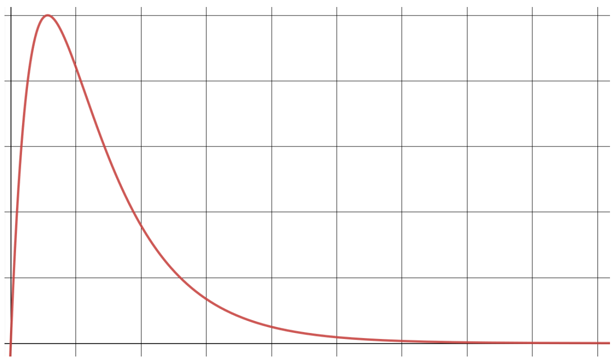


Figure 4.5: Qualitative plot of the function $Ae^{-Bt}(1 - e^{-Ct})$

This will give a function for the motor command:

$$\tau(s) = AC\frac{s + k_0}{s(s + B)(s + B + C)} \tag{4.12}$$

And a function for the yaw:

$$\theta_y(s) = \tau(s)H(s) = ACK\omega_0\frac{s + k_0}{s(s + B)(s + B + C)(s + \omega_0)} \tag{4.13}$$

$$\begin{aligned}\theta_y(t) = ACK\omega_0(\frac{k_0}{B(B + C)\omega_0} + \frac{k_0 - B}{BC(B - \omega_0)}e^{-Bt} + \frac{k_0 - (B + C)}{(B + G)C(\omega_0 - (B + C))}e^{-(B+C)t} + \\ \frac{\omega_0 - k_0}{\omega_0(B - \omega_0)(B + C - \omega_0)}e^{-\omega_0 t}) + \theta_0\end{aligned} \tag{4.14}$$

As mentioned on the previous variant A won't be the parameter trained but instead it will be the parameter D directly connected to the final motor position. To shortcut the expression that relates A and D, the Final Value Theorem [42] can be used:

$$\lim_{t\to\infty} \tau(t) = \lim_{s\to 0} s\tau(s) \Rightarrow D = \frac{ACk_0}{B(B + C)} + F \tag{4.15}$$

As it can be seen there is this new parameter C across all expressions introduced by the function C. This C represents the rate of increase of the step $f(t)$, the higher this value the faster will be the step, and to avoid making this parameter another one that has to be trained by the algorithm, this value will be constrained: C will have a value such that the maximum velocity of the motor won't rise above a certain value $\mathcal{V}$:

$$\tau(t) = AC(\frac{k_0}{B(B+C)} + \frac{B-k_0}{BC}e^{-Bt} + \frac{k_0-(B+C)}{(B+C)C}e^{-(B+C)t}) \tag{4.16}$$

$$\frac{d\tau}{dt} = AC(-\frac{B-k_0}{C}e^{-Bt} - \frac{k_0-(B+C)}{C}e^{-(B+C)t}) \leq \mathcal{V} \tag{4.17}$$

Since the maximum will happen at $t = 0$:

$$AC \leq \mathcal{V} \tag{4.18}$$

There is two ways to go about the problem raised by the expression above: C is going to be a parameter that will vary depending on the value of A, this is, for every saccade the maximum velocity of the motor will be $\mathcal{V}$, or C would be kept constant independently of the saccade size - the maximum velocity would only be achieved for the longer saccades (highest A). Both of the options will be evaluated and the value of $\mathcal{V}$ will be 7200, corresponding to a top speed of 20 rotations per second.

4.3.3 Variant 3

This last variant will tackle an issue that causes the model to be unrealistic in both of the previous variants: $\frac{d\tau}{dt}|_{t=0} > 0$ which implies that the motor doesn't start in steady state. Therefore $f(t)$ has to be a function that has a first derivative equal to 0: such function will be a second order critically damped step $f(t) = (1 - e^{-Ct} - Cte^{-Ct})$.

$$p_m(t) = Ae^{-Bt}(1 - e^{-Ct} - Cte^{-Ct}) \tag{4.19}$$

$$p_m(s) = \frac{AC^2}{(s+B)(s+B+C)^2} \tag{4.20}$$

Which looks like this:

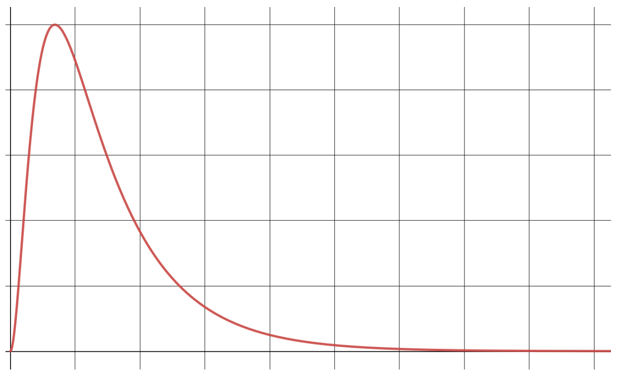


Figure 4.6: Qualitative plot of the function $Ae^{-Bt}(1 - e^{-Ct} - Cte^{-Ct})$

Making $\tau(s)$:

$$\tau(s) = AC^2 \frac{s+k_0}{s(s+B)(s+B+C)^2} \tag{4.21}$$

And so the yaw θ_y:

$$\theta_y(s) = \tau(s)H(s) = AC^2 K\omega_0 \frac{s+k_0}{s(s+B)(s+B+C)^2(s+\omega_0)} \tag{4.22}$$

$$\begin{aligned}\theta_y(t) = AC^2K\omega_0(&\frac{k_0}{B(B+C)^2\omega_0} + \frac{k_0 - B}{-BC^2(\omega_0 - B)}e^{-Bt} + \frac{-(B+C)^2 + k_0(B+2C)}{C^2(B+C)^2}e^{-(B+C)t} \\ &+ \frac{k_0 - (B+C)}{(B+C)C(\omega_0 - (B+C))}te^{-(B+C)t} + \frac{k_0 - \omega_0}{-\omega_0(B-\omega_0)(B+C-\omega_0)^2}e^{-\omega_0 t}) + \theta_0\end{aligned} \tag{4.23}$$

Proceeding the same way as before to find an expression in terms of D:

$$\lim_{t\to\infty} \tau(t) = \lim_{s\to 0} s\tau(s) \Rightarrow D = \frac{AC^2k_0}{B(B+C^2)} + F \tag{4.24}$$

And lastly, the same logic as the previous variant will be done to define the parameter C. Though, in this case the way to find a value for this parameter is less trivial. And so a different approach will be taken to define:

$$\frac{d\tau}{dt} \leq \mathcal{V} \tag{4.25}$$

Firstly $\tau(t)$ will be separated into its pulse $p_m(t)$ and step functions $s_m(t)$:

$$\tau(t) = p_m(t) + s_m(t) \tag{4.26}$$

$$s_m(t) = Ak_0(\frac{C^2}{B(B+C)^2} - \frac{e^{-Bt}}{B} + \frac{1 + Ct + C(B+C)}{(B+C)^2}e^{-(B+C)t}) \tag{4.27}$$

And realize that:

$$p_m(t) \leq A(1 - e^{-Ct} - Cte^{-Ct}) \tag{4.28a}$$

$$s_m(t) \leq \frac{Ak_0C^2}{B(B+C)^2}(1 - e^{-Bt}) \tag{4.28b}$$

And:

$$\frac{dp_m(t)}{dt} \leq \frac{d}{dt}(A(1 - e^{-Ct} - Cte^{-Ct})) \leq max(\frac{d}{dt}(A(1 - e^{-Ct} - Cte^{-Ct}))) = \frac{AC}{e} \tag{4.29a}$$

$$\frac{ds_m(t)}{dt} \leq \frac{d}{dt}(\frac{Ak_0C^2}{B(B+C)^2}(1 - e^{-Bt})) \leq max(\frac{d}{dt}(\frac{Ak_0C^2}{B(B+C)^2}(1 - e^{-Bt}))) = \frac{Ak_0C^2}{(B+C)^2} \tag{4.29b}$$

As such it can be constrained:

$$\frac{d\tau(t)}{dt} = \frac{dp_m(t)}{dt} + \frac{ds_m(t)}{dt} \leq \frac{d}{dt}(A(1 - e^{-Ct} - Cte^{-Ct})) + \frac{d}{dt}(\frac{Ak_0C^2}{B(B+C)^2}(1 - e^{-Bt})) \tag{4.30}$$

And so from equations 4.25, 4.29 and 4.30 it can be said that:

$$\frac{d\tau(t)}{dt} = \frac{dp_m(t)}{dt} + \frac{ds_m(t)}{dt} \leq \frac{AC}{e} + \frac{Ak_0C^2}{(B+C)^2} \leq \mathcal{V} \tag{4.31}$$

And so C is constrained to $\frac{AC}{e} + \frac{Ak_0C^2}{(B(B+C)^2)} \leq \mathcal{V}$ which is a solvable expression that will define an upper bound for C.

Figure 4.7 demonstrates the architecture of the policy network which will should give a better insight to the relationship between the state values as an input to the action values as an output: And for further

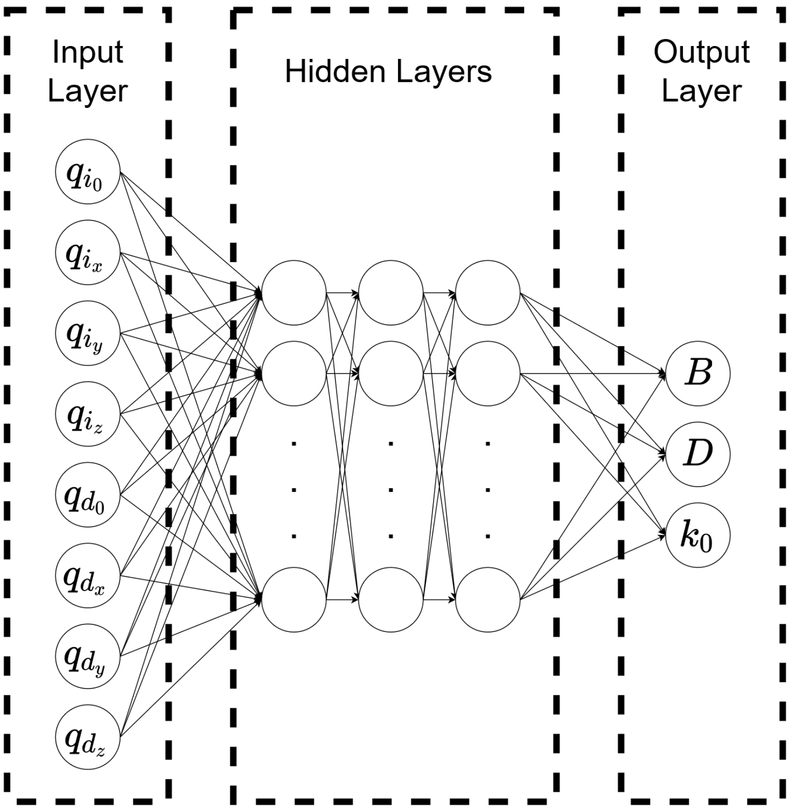


Figure 4.7: Policy network architecture relating the state value to the action value

clarification, the output layer will feed into the pulse+step generator as described in Figure 4.8, to form a motor command $\tau(t)$ that will feed into the eye plant:

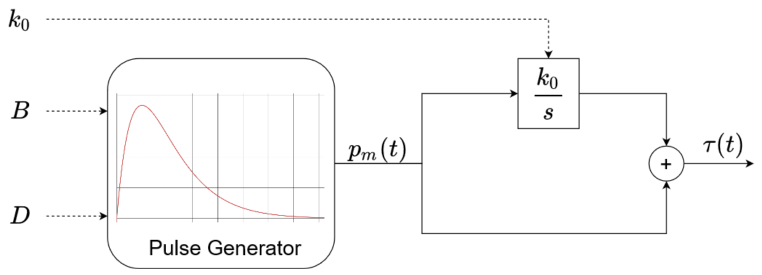


Figure 4.8: Pulse+step generator with B, D and k_0 as inputs

4.4 Reward

The reward function takes into account costs (penalties) on lack of accuracy, energy spent and saccade duration, as in previous works [3–7], to which we add overshoot introduced by parameter k_0, which is an undesired and atypical property. In reinforcement learning, it is intended to maximize the value of the reward, and so the negative of these 4 costs will define the reward:

$$R_{total} = \lambda_a R_a + \lambda_e R_e + \lambda_d R_d + \lambda_o R_o \tag{4.32}$$

where R_{total} corresponds to the total reward, R_a corresponds to the accuracy reward, R_e corresponds to the energy reward, R_d corresponds to the duration reward and finally R_o is the overshoot reward. The λs are weights that will control the relevance of each of the rewards - depending on the values of these the optimal solutions obtained after training would be different.

4.4.1 Energy Reward

In previous works, the energy cost J_e would be the sum of the squared differences of consequent motor positions Δu:

$$J_e = \sum_{i=1} \frac{(\Delta u_i)^2}{\Delta t_i} \tag{4.33}$$

This was applied in discrete optimal control, but since on this work there are analytical solutions for the motor command τ, the equation 4.33 can be transformed to a continuous domain:

$$J_e = \sum_{i=1} \frac{(\Delta u_i)^2}{(\Delta t_i)^2} \Delta t_i \Rightarrow \lim_{\Delta t_i \to 0} \sum_{i=1} \left(\frac{\Delta u_i}{\Delta t_i} \right)^2 \Delta t_i = \int \left(\frac{d\tau}{dt} \right)^2 dt \tag{4.34}$$

And so the energy reward will be:

$$R_e = - \int \left(\frac{d\tau}{dt} \right)^2 dt \tag{4.35}$$

This will lead to different energy functions for each of the variants. So for variant 1:

$$R_e = -A^2 \left[\left(\frac{1 - 2(B-k_0)t + (B-k_0)^2 t^2}{-2B} + \frac{-2(B-k_0) + 2(B-k_0)^2 t}{-4B^2} + \frac{2(B-k_0)^2}{-8B^3} \right) e^{-2Bt} \right]_0^\infty \tag{4.36}$$

For variant 2:

$$R_e = -A^2 \left[\frac{(k_0 - B)^2}{-2B} e^{-2Bt} + \frac{2(k_0 - B)(B+G-k_0)}{-(2B+G)} e^{-(2B+G)t} + \frac{(B+G-k_0)^2}{-2(B+G)} e^{-2(B+G)t} \right]_0^\infty \tag{4.37}$$

And finally for variant 3:

$$\begin{aligned} R_e = &-A^2 C^4 \left[\frac{a_0^2}{-2B} e^{-2Bt} - \left(\frac{2a_0(a_1 + a_2 t)}{-(2B+C)} - \frac{(2a_0 a_2)}{(2B+C)^2} \right) e^{-(2B+C)t} \right]_0^\infty \\ &-A^2 C^4 \left[\left(\frac{(a_1 + a_2 t)^2}{-2(B+C)} - \frac{2a_2(a_1 + a_2 t)}{4(B+C)^2} - \frac{2a_2^2}{8(B+C)^3} \right) e^{-2(B+C)t} \right]_0^\infty \end{aligned} \tag{4.38}$$

where $a_0 = \frac{k_0 - B}{C^2}$, $a_1 = \frac{B - k_0}{C^2}$ and $a_2 = \frac{B+C-k_0}{C}$.

Even though in the human eye this term can be considered inconsequential due to the many saccades a person performs in a day [43], in this work this term is important because it will be the one used to balance between all the terms, for example, if the goal was to perform the fastest saccades possible disregarding energy costs, intuitively by looking at the equations of $\tau(s)$ and subsequently $\theta_y(d)$ for the different action variants the solution would be trivial: k_0 would have the same value as the constant of the plant, B would infinitely large, and A (or D) would be ones that would match the desired saccade size.

4.4.2 Duration Reward

The expression for duration reward will be based on the work of Shadmehr [44], where he found the duration cost J_d of a saccade in humans follows a hyperbolic function:

$$J_d = 1 - \frac{1}{1 + \beta t_d} \tag{4.39}$$

where t_d corresponds to the duration of a saccade and β is the temporal discount rate - which will have a value of 0.6 [45]. The value t_d that's going to be obtained from applying the motor commands to the plant, will be the settling time of 1%, as in, it will be the time taken for the eye to reach an orientation that is 1% of its final value. This has to be this way because since we are dealing with continuous exponential functions, the time which the eye will take to reach exactly its final position will be ∞. Moreover, when moving to the simulator or the real model there will be noise and drift on the measurement of the orientation of the eye due to imperfection on the sensors, possible slack for the elastics, or residual velocities [5, 7], making this a better metric for duration. So the reward function for duration will simply be.

$$R_d = -(1 - \frac{1}{1 + 0.6 t_d}) \tag{4.40}$$

Intuitively this term is necessary make faster saccades.

4.4.3 Accuracy Reward

The accuracy cost is the squared difference between the target saccade direction and the the final saccade orientation. In quaternions, the representation of the reward of this expression is [46]:

$$R_a = -(1 - (q_d \cdot q_f)^2) \tag{4.41}$$

Where q_d corresponds to the desired/ target quaternion orientation, q_f corresponds to the final quaternion orientation of the eye and $\cdot$ correponds to the inner product of the two quaternions. Similarly to what was done in 4.4.2 and for the same reasons, this final eye orientation will actually be the orientation of the eye at the settling time - this will also have the extra advantage for determining the values of the weights of each of the reward terms which will be explained in 4.4.5.

4.4.4 Overshoot Reward

Even though in previous works there is no need for the introduction of this term, on this one it is necessary because it is a saccade property that is introduced by the term k_0. Besides it having some influence in the duration and energy cost of the saccade, it majorly affects the overshoot, and since this is something undesired in a saccade there's a need for punishment when this happens. The overshoot is the maximum peak value of a signal measured relative to its steady state response [47] and it will be specially necessary to take into account when the response has an overshoot with peak value within its settling time error band - since saccade tend to have an undershooting bias [48], because intuitively, an overshot saccade has a greater expenditure of energy. Therefore using the current metric to measure duration, this type of small overshoot will lead to a lower duration time. With a low energy weight and without this overshoot, this would lead to solutions of saccades that would maximize the overshoot within the settling time error band:

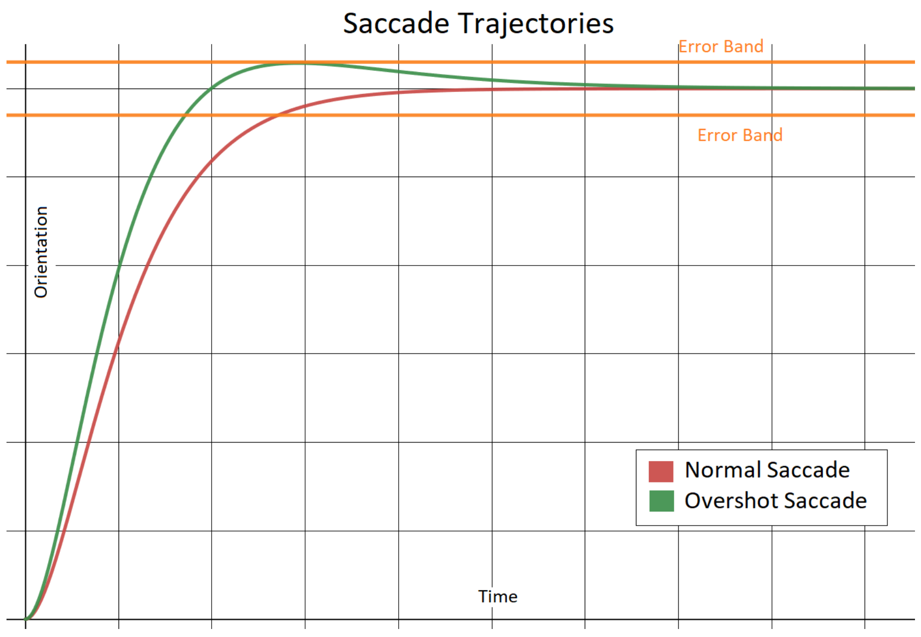


Figure 4.9: Comparison between a normal saccade with an overshot saccade trajectory

Figure 4.9 illustrates exactly this, the overshoot has a lower settling time, because its peak is within the error band, and therefore it would have had a higher reward if overshoot was unaccounted for.So, for a saccade that has initial yaw θ_0, a final yaw of θ_f and maximum yaw of θ_{peak}, the equation for the overshoot reward would be:

$$R_o = -\left(\frac{\theta_{peak} - \theta_0}{\theta_f - \theta_0} - 1\right) \tag{4.42}$$

4.4.5 Weights

The value of the weights for each of the rewards is important to get to a desired solution. For instance, the value of each of the rewards will always a value less than zero, and so theoretically the maximum value for the total reward is zero, and so a trivial solution to get the maximum value of the total reward would be for each of the λs to be zero - this would mean, though, that there would be no relationship between the inputs and outputs of the system, and therefore no learning would be possible. One way to

get values for these weights would be to grab values based on literature, and apply them to the model and verify if they work. With this method, however, the only thing that it would be possible to verify is if the algorithm learns, or not, to make desired saccades. There would be no way to verify besides exhaustive searching, whether or not, the solution is the optimal solution, nor, even if all this information was known, it would be possible to control if the solution is the one desired for the model. The method that it is going to implemented will deal with both of these issues: from the beginning the optimal policy for one specific saccade will be selected and the weights will be calculated from that solution. The algorithm will then be trained with those weights and if the algorithm learns and reaches the same solution for that specific saccade, it can also be assumed that the optimal policy was learned for all saccades.
The method will be the following: for a desired saccade the actor network will output 3 parameters, B, D (or A) and k_0 which will be part of a motor command τ, which will move the plant and consequently output a reward, so, in essence, the reward is a function of these parameters, $R_{total}(B, D, k_0)$, and as the individual rewards are calculated individually and then added together, they can be considered as functions of the 3 parameters as well ($R_a(B, D, k_0)$, $R_e(B, D, k_0)$, $R_d(B, D, k_0)$, $R_o(B, D, k_0)$). And so, basically:

$$R_{total}(B, D, k_0) = \lambda_a R_a(B, D, k_0) + \lambda_e R_e(B, D, k_0) + \lambda_d R_d(B, D, k_0) + \lambda_o R_o(B, D, k_0) \tag{4.43}$$

And so a condition for it R_{total} to be an optimum value is for the value of its derivative to be zero for all parameters:

$$\nabla R_{total}(B, D, k_0) = 0 \tag{4.44}$$

And so:

$$\begin{aligned}
\nabla R_{total}(B, D, k_0) &= \nabla(\lambda_a R_a(B, D, k_0) + \lambda_e R_e(B, D, k_0) + \lambda_d R_d(B, D, k_0) + \lambda_o R_o(B, D, k_0)) \\
\nabla R_{total}(B, D, k_0) &= \nabla \begin{bmatrix} R_a(B, D, k_0) & R_e(B, D, k_0) & R_d(B, D, k_0) & R_o(B, D, k_0) \end{bmatrix} \begin{bmatrix} \lambda_a \\ \lambda_e \\ \lambda_d \\ \lambda_o \end{bmatrix} \\
0 &= \nabla \begin{bmatrix} R_a(B, D, k_0) & R_e(B, D, k_0) & R_d(B, D, k_0) & R_o(B, D, k_0) \end{bmatrix} \begin{bmatrix} \lambda_a \\ \lambda_e \\ \lambda_d \\ \lambda_o \end{bmatrix}
\end{aligned} \tag{4.45}$$

That system of equations seems to be unsolvable, there is an infinite number of solutions. Since, intuitively, what matters technically is the ratio between each of the weights the optimal - the optimal solution for the total reward function with all weights equal to one should be the same as if all the weights would be 2, or 3, the only thing that would change would be the scale of the rewards. Knowing this, one of the solutions would be to use a method with Langrange multipliers, with a constraint that would fix all of the weights to the hypersphere, or similarly, something that yeilds the same results - fix one of the values. since as mentioned before energy tends to be the most unimportant term, this will be fixed to one:

$$0 = \nabla \begin{bmatrix} R_a(B, D, k_0) & R_e(B, D, k_0) & R_d(B, D, k_0) & R_o(B, D, k_0) \end{bmatrix} \begin{bmatrix} \lambda_a \\ 1 \\ \lambda_d \\ \lambda_o \end{bmatrix} \tag{4.46}$$

Which now becomes possibly solvable with one solution:

$$\begin{gathered} -\nabla R_e(B, D, k_0) = \nabla \begin{bmatrix} R_a(B, D, k_0) & R_d(B, D, k_0) & R_o(B, D, k_0) \end{bmatrix} \begin{bmatrix} \lambda_a \\ \lambda_d \\ \lambda_o \end{bmatrix} \\ \begin{bmatrix} -\frac{dR_e(B,D,k_0)}{dB} \\ -\frac{dR_e(B,D,k_0)}{dD} \\ -\frac{dR_e(B,D,k_0)}{dk_0} \end{bmatrix} = \begin{bmatrix} \frac{dR_a(B,D,k_0)}{dB} & \frac{dR_d(B,D,k_0)}{dB} & \frac{dR_o(B,D,k_0)}{dB} \\ \frac{dR_a(B,D,k_0)}{dD} & \frac{dR_d(B,D,k_0)}{dD} & \frac{dR_o(B,D,k_0)}{dD} \\ \frac{dR_a(B,D,k_0)}{dk_0} & \frac{dR_d(B,D,k_0)}{dk_0} & \frac{dR_o(B,D,k_0)}{dk_0} \end{bmatrix} \begin{bmatrix} \lambda_a \\ \lambda_d \\ \lambda_o \end{bmatrix} \end{gathered} \tag{4.47}$$

One of the conditions for failure on the equation 4.47 would be for any of the columns have all derivatives equal to zero meaning one of two things: either the change in any of the parameters has no effect on the value of that reward and so the weight of that reward would be zero, or the function of that reward is at a local optimum, making the importance of that reward infinitely high. And so, as mentioned before, if the accuracy reward were to be done with the exact final position, the derivative would've been 0 for all the parameters.

So, now with this method, it is possible to pick the optimal policy for one specific saccade, obtain the weights, and let the algorithm train the model with those weights. The values of parameters chosen for the optimal policy will be $B = 35$, twice the size of the eye model, $D \approx 20.33$ a value for the final position of the motor, corresponding to $30\ ^o$ of yaw as the final eye orientation ($D \times K = 30$), and $k_0 = 17.5$, corresponding to the value of the plant. This is, if we let the algorithm train, for a desired yaw of $30\ ^o$ the policy network should output the parameters $B = 35$, $D = 20.33$ and $k_0 = 17.5$.

Something non-trivial to obtain might be the equations of each of the individual rewards with respect to the parameters, however this issue can be easily solved by applying a finite differences method, for values around the desired solutions, as in, obtain the value of each individual rewards around the interest points and use numerical differences to obtain approximations of the derivatives.

4.5 Overview

Figure 4.10 shows a diagram of the implementation of the algorithm and how its different components interact with each other.

For each training episode, an initial and desired orientation are randomly selected, which pass has a state vector to the policy network, generating an action, a value for B, D and k_0 (Figure 4.7). This action

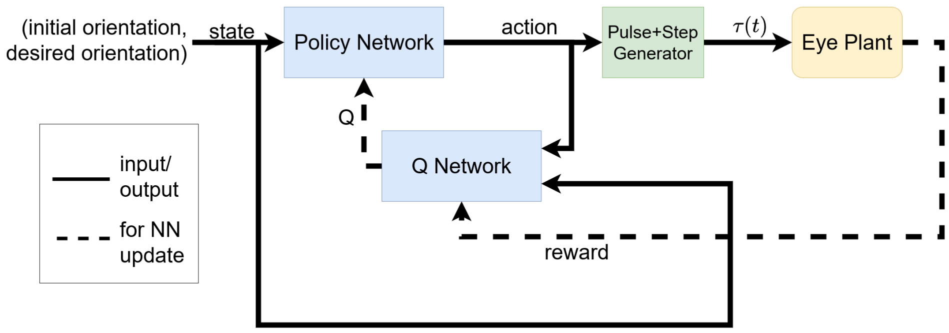


Figure 4.10: Overview of the implementation of the algorithm.

is converted to a motor movement through the pulse+step generator (Figure 4.8), and it is applied to the eye plant, outputting a reward after the eye movement is concluded. The Q network evaluates the action applied to the state through a Q value (an approximation of the reward) and uses it to improve the policy network.

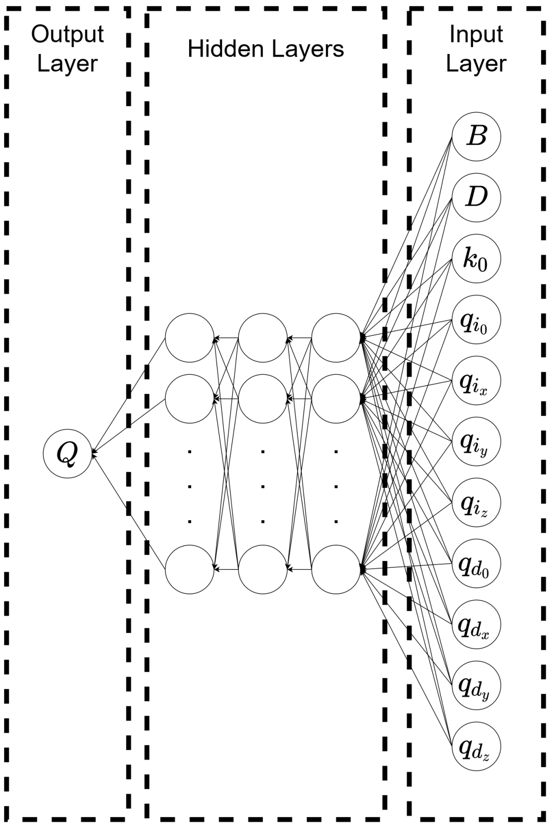


Figure 4.11: Q Network Architecture

4.6 Further Remarks

This part will be dedicated mainly to explain and justify some of the hyperparameter tuning and specific training aspects. Some of these values were simply taken from advice from some machine learning literature and others just by trial and error.
Starting with the neural networks for the policy network, for the Q-function network and for the target value network. The policy network and the Q-function will be multilayer perceptron (MLP), with 3 hidden layers, the first two with a size of 128 neurons each, and the last one with 32 neurons, all with an activation function of a linear rectifier which indicated for regression [49]. For the target value network, there won't be a need for a MLP since the output of this should technically be always 0, since the control is done in open-loop, once the eye reaches is final position, which would be the final state, there's no extra actions to be applied to the model and consequently there will be no further rewards to be obtained. It can be argued that maybe the size of the neural networks may be to large for the problem at hand, but since the trainings were done quickly, each taking about 6 hours without issues, there was no need to decrease these values.
The learning rates for the neural networks, will be set to 10^{-5} because it was tested to work fine and they go well with a sizeable batch size for training of 8196 [50]. Moreover, the Adam optimizer [51] was chosen to make the updates on the neural networks, since it is considered as the most stable and robust optimizer amongst many others [52, 53].
Finally the training is done in a span of 300 thousand episodes, for saccades that have a random starting position, and a random desired target orientation, both within -35 and 35 ^{o}, and with replay pool of 30 thousand instances. As a final note, all of these hyperparameters mentioned could be fine tuned to achieve faster results, not only timewise but also in the number of episodes needed train the networks, but doing so to a model that served as proof of concept was perceived as excessive. This whole implementation was developed in python with the help of the Tensorflow library to build the Soft Actor-Critic algorithm, and the training was done on a machine using Ubuntu 20.04.2 LTS operating system, with an Intel(R) Core(TM) i7-9700F CPU@3.00GHz and a GeForce RTX 2700 Super graphics card.

Chapter 5

Results

This chapter aims to present the results obtained from using the soft actor critic algorithm to create a open-loop control policy for the saccadic model using the implementation explained in the previous chapter. The model will be trained for 3 different types of models for action which will define the motor commands applied to the model. The training time for each of the action types will be done for a total of 300 thousand episodes and with reward weights such that when a $30\ ^o$ saccade is desired, the output of the policy network should be on average: $B = 35$, $k_0 = 17.5$ and $D = 20.33$.
There will be various components that will be used to validate the results obtained. Firstly for each of the action variants the average output of each of the parameters will be plotted against the saccade size desired and be compared with an estimation of the optimal output. This estimation will be obtained through a search of the best reward on the region of interest - all the space that the algorithm had access to explore. In this context the average output will be consider, that the output will be the mean of all the possible saccades with a certain size, for example, for a $30\ ^o$ saccade size this will include the policy output from a orientation of $0\ ^o$ to a $30\ ^o$ orientation, all the way to a starting orientation of $-30\ ^o$ all the way to $0\ ^o$, including everything in between. Since D describes the final position of the motor, it doesn't make sense to show the average of these values, instead it would be displayed the mean of the amplitude between the final and initial motor position ($D - F$). After that, it will be displayed a visualization of the pulse, pulse-step signal and saccade trajectory when the policy obtained is applied to the model for different saccade sizes. Along with the main sequences, and skewness plots for those saccades it will be possible to compare with expected qualities in human data and validate the type of action taken.

5.1 Variant 1

Off the three variants for the action this one is the simpler one: the pulse shape of the command is $p_m(t) = Ate^{-Bt}$. From the equation 4.47 it was obtained the following values for the weights: $\lambda_a = 3.8 \times 10^6$, $\lambda_e = 1.0$, $\lambda_d = 4.9 \times 10^4$ and $\lambda_o = 2.3 \times 10^8$. By looking at the weights it can be expected resulting accurate saccades, fast saccades with no overshoot. Figure 5.5 shows the average output policy parameters against the desired saccade size:

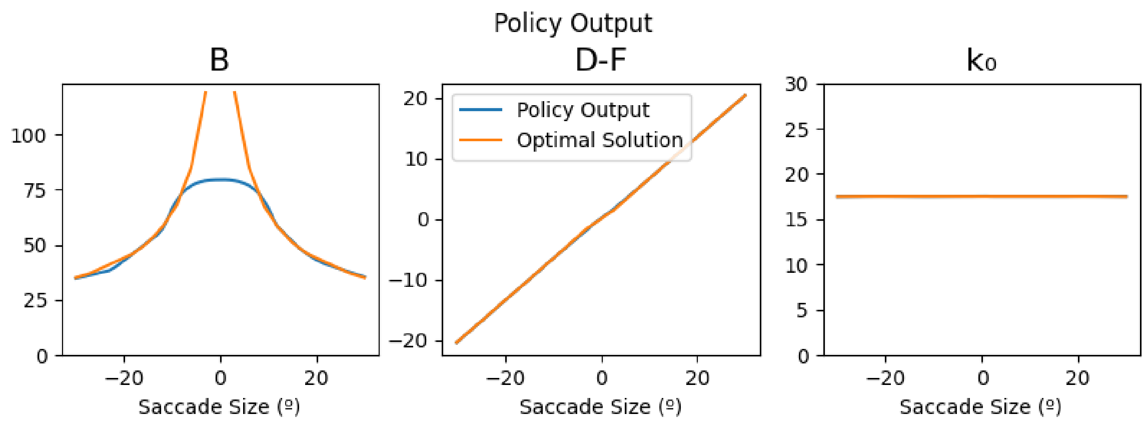


Figure 5.1: Policy output of the parameters against the desired saccade size of variant 1

As it can be seen in figure 5.1, there is a very close relationship between the estimated optimal policy and the output policy of the actor. Both for the amplitude between the final motor position (D) and the initial motor position (F), and k_0 it can be confirmed the linearity of the model: the motor amplitude $(D-F)$ shows a proportional relationship between itself and the saccade amplitude, with a value of around 1.5 which matches the value used for K (1.4758) in the plant. k_0 shows a constant relationship, which is exactly what is expected, it matches the cut-off frequency of the plant. If a non-linear model were to be applied using this type of pulse it could be prematurely assumed that both these parameters would match the non-linearity of the plant. Regarding the parameter B, this is where the non-linearity of the control system is introduced; the policy has a higher value for this parameter, the lower the saccade amplitude. Intuitively as the desired saccade size would tend to zero, this value would infinitely increase, and that's the root cause of the discreapancy between the policy output and the expected optimal output for the shorter saccades. There is a saturation of parameter B, due to the upper bound defined for it before training, which was 80. If a higher upper bound were to be defined, there would be a higher matchup between the estimated optimal policy and the actor output, however for the lowest saccade amplitudes this saturation, though smaller, would still be present.
The next graph will show the pulse signal generated, subsequent pulse step signal applied to the model, and the saccade trajectories that were output.

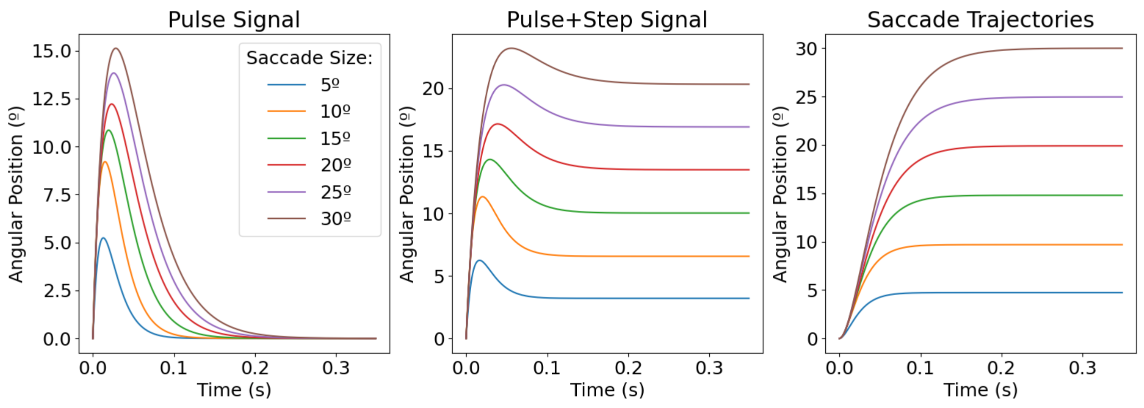


Figure 5.2: Pulse Signal, Pulse+Step Signal and Saccade trajectory output employed by the policy network of variant 1

As seen the expected and wanted behaviour is present in the 3 graphs for the different saccade amplitudes. The peak of the pulse increases with the saccade size, as well as the amplitude of the step, and, as it can be observed, all these signals make the saccade have a step like response to the desired orientation, or at the very least, very close to it. Moreover when compared to the output of a pure step input:

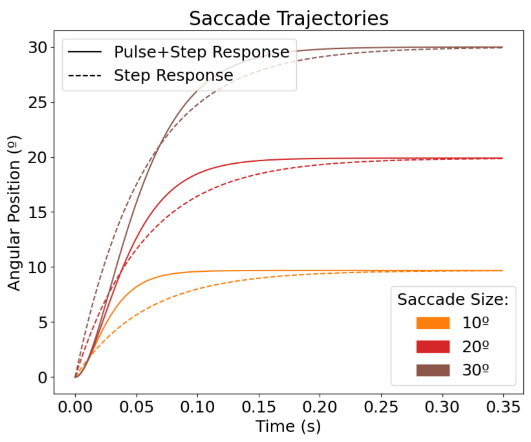


Figure 5.3: Comparison of the response between a pulse+step signal to a pure step signal for variant 1

it can clearly be seen, as expected, the saccades produced with a pulse+step signal are faster, but with a slower response at the start. Also it's just shown three saccade results to avoid clustering.
Now, by looking at the main sequence and skewness properties it will be possible to compare the properties of the saccades retrieved to the human data.

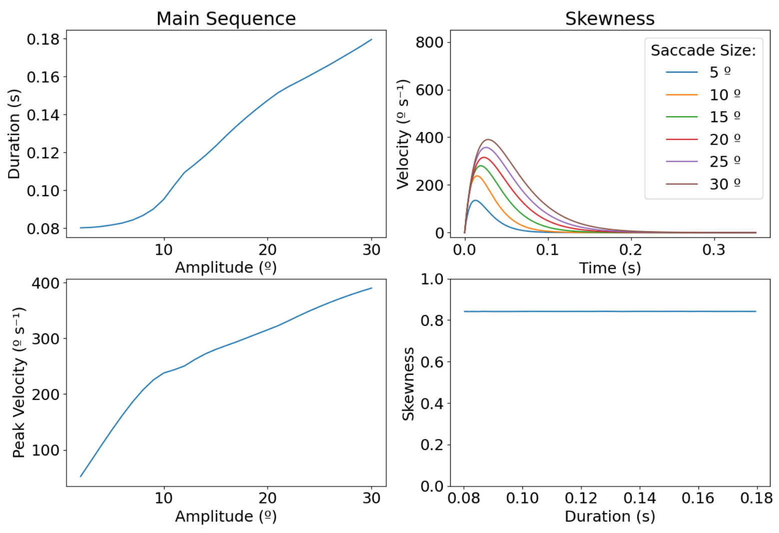


Figure 5.4: Main sequence and skewness of the saccades obtained of variant 1

As it can be seen on the left part of figure 5.4, the properties obtained seems to follows partially the same pattern has the ones found in humans - on one hand there is an increase in duration with the increase in saccade amplitude and there seems to be a non-linear relationship between peak velocity and amplitude. In both cases the inconsistency in the smaller amplitudes might be a consequence of the saturation on the value of B for smaller saccades. On the other hand, on the right side, there seems to be an increase in skewness on the top graph, although this doesn't seem to be reflected on the bottom graph: one of the reasons for this is due to the precision and how this value is calculated - the ratio between the duration of the saccade and the deceleration time. Upon a zoomed in inspection on that graph it is seen that the output for the skewness is very noisy.

5.2 Variant 2

This is one of the variants that will have the introduction of another parameter (C) in the pulse signal: $p_m(t) = Ae^{-Bt}(1 - e^{-Ct})$. As such there is the need to divide the problem into 2 possible scenarios: the scenario where the value of C is fixed or the scenario where C varies such that the motor always outputs its maximum defined velocity at some point during the saccade execution. This is, instead of defining a value of C at the beginning, and using it for every episode, C is calculated every time the policy network outputs an action, in this case using the constraint in equation (4.18).

5.2.1 Fixed C

For the fixed C instance, this value is calculated by applying the equation 4.18 for the larger values of saccade sizes. For a $30\ ^o$ saccade it is desired for the motor output its maximum velocity then, knowing the desired values of B, D and k_0, and applying equation 4.15 it is possible to obtain A and therefore a value of $C = 85$. With this, the value for the weights of the rewards calculated are: $\lambda_a = 1.7 \times 10^8$, $\lambda_e = 1.0$, $\lambda_d = 2.3 \times 10^6$ and $\lambda_o = 3.6 \times 10^8$. Even though they have different values and orders of magnitude it could be safely expected a similar behaviour to what happened in the previous example. Figure 5.5 shows the average output policy parameters against the desired saccade size:

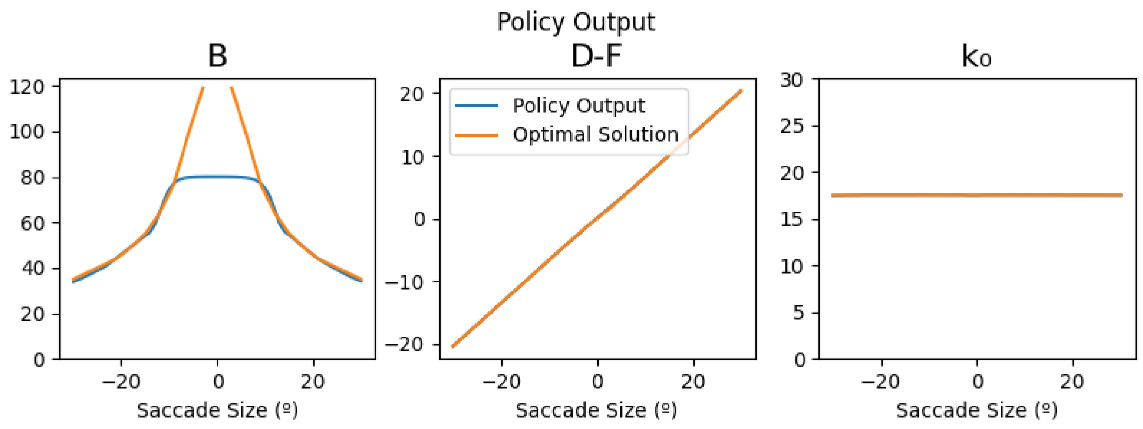


Figure 5.5: Policy output of the parameters against the desired saccade size of variant 2 with fixed C

The graph shows above the policy output for this second variant of the pulse signal with a fixed value of C. Similar behaviour as the previous variant can be observed in Figure 5.5: D and k_0 follow the expected linear behaviour and B follows the non-linear behaviour, where it gets larger the smaller the saccade amplitude and it presents its saturation from the learning constraints.

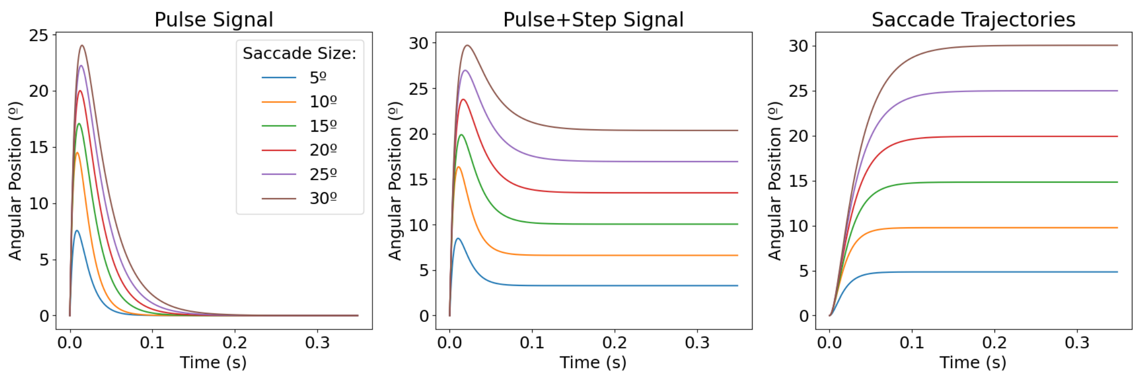


Figure 5.6: Pulse Signal, Pulse+Step Signal and Saccade trajectory output employed by the policy network of variant 2 with fixed C

In contrast with the results of the previous variant, the signals appear to have a higher peak and a shorter duration resulting in overall faster saccade trajectories, which is due to naturally higher slope values for the raising part of the pulse signal. As such B will have to be higher to compensate for higher peak velocities - in figure 5.5 there seems to be a higher saturation region when compared to Figure 5.1.

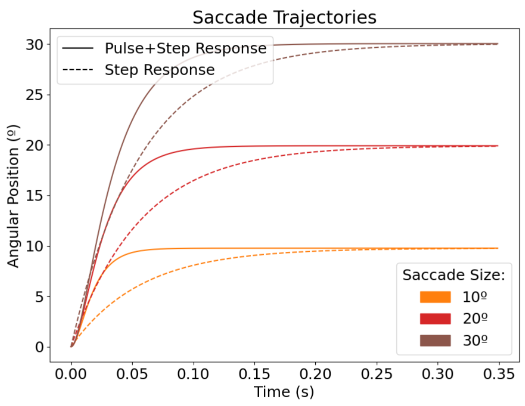


Figure 5.7: Comparison of the response between a pulse+step signal to a pure step signal for variant 2 with fixed C

Nonetheless, it can be seen in Figure 5.7 that the pulse+step signal still produces a much faster saccade compared with a pure step as an input signal.

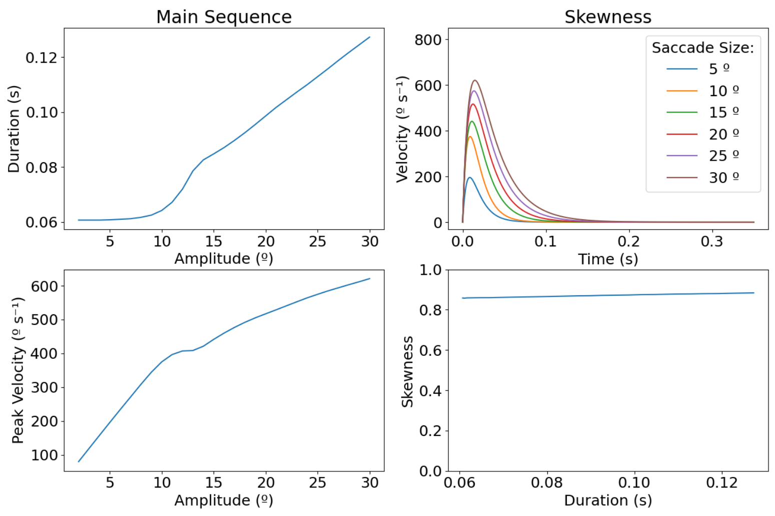


Figure 5.8: Main sequence and skewness of the saccades obtained of variant 2 with fixed C

Just like what was presented in figure 5.4, the figure 5.8 exhibits a similar behaviour. There is the non-linear relationship between the saccade amplitude and the duration, and between the peak velocity and amplitude, and a apparent linear relationship between skewness of the velocity profiles and duration. Again inconsistencies of these values for lower saccades amplitudes and duration is due to the saturation of B during training.

5.2.2 Varying C

For this instance the values obtained for the weights through equation 4.47 are: $\lambda_a = 1.2 \times 10^8$, $\lambda_e = 1$, $\lambda_d = 1.3 \times 10^6$ and $\lambda_o = 1.8 \times 10^8$ - as these have similar order of magnitude it could be expected similar results for the output parameters: As well as C, since its movable:

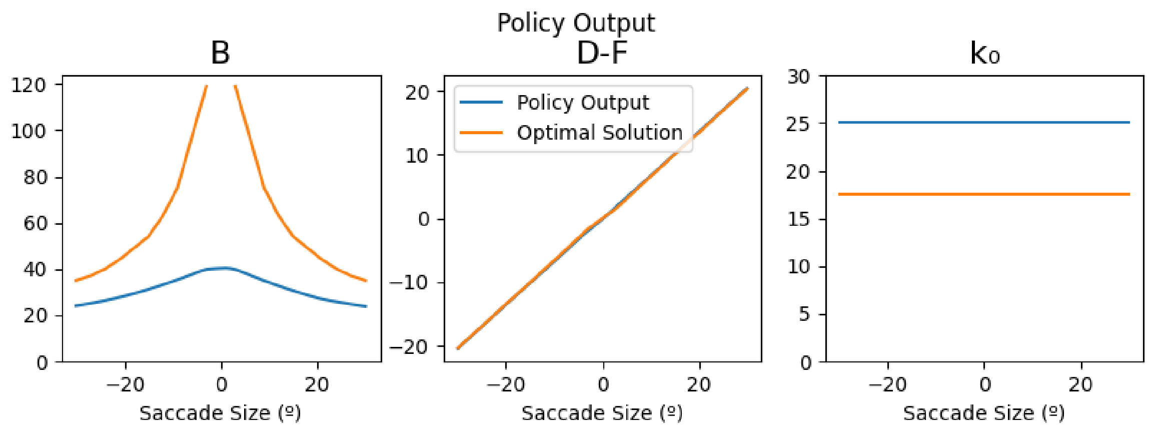


Figure 5.9: Policy output of the parameters against the desired saccade size of variant 2 with varying C

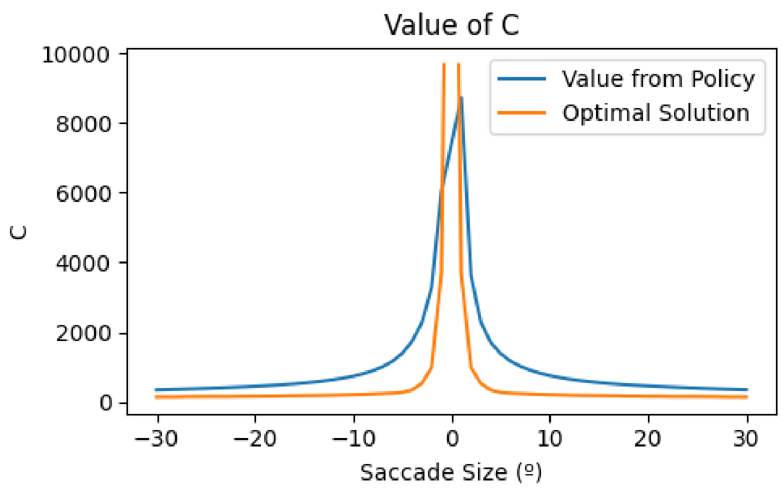


Figure 5.10: Value of C with Saccade Size for variant 2

Unfortunately this was not the case, there is a mismatch between what would be expected and the results obtained. The D parameter is the only one to follow the linear match with the plant, B has a lower policy output when compared with the optimal solution, even though it is completely off target for set optimal value of 35 for a $30\ ^o$ saccade. Likewise, for the value of k_0, the obtained value is completely off the target of 17.5 for the entire range of saccades, meaning that there isn't a cancellation anymore between the gain of the integrator and the constant of the plant (equation 4.12). As a consequence there's also the mismatch between the expected value of C and the value of C obtained. It is expected then, that the output saccades will be much slower compared with what previously obtained:

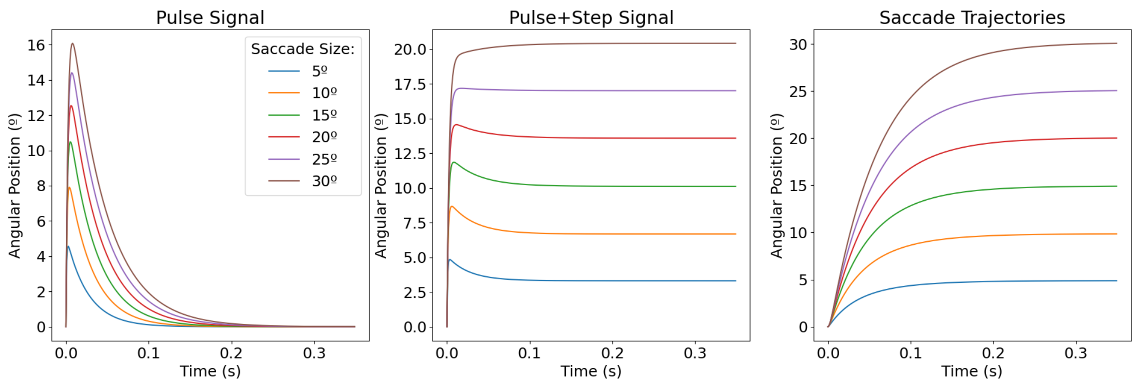


Figure 5.11: Pulse Signal, Pulse+Step Signal and Saccade trajectory output employed by the policy network of variant 2 with varying C

As foreseen, this is exactly what happened. The pulse plus step signals convey, specially for higher saccade amplitudes, just a step like function - the presence of the pulse is minimal. As such it is expected that the response of the model could be approached to just its pure step response:

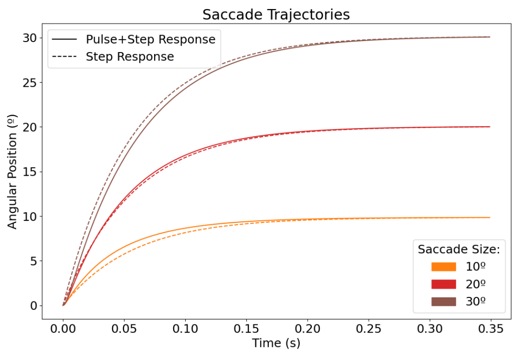


Figure 5.12: Comparison of the response between a pulse+step signal to a pure step signal for variant 2 with varying C

And as seen above, figure 5.12, that is exactly what happens.

Now looking at the main sequence properties and skewness:

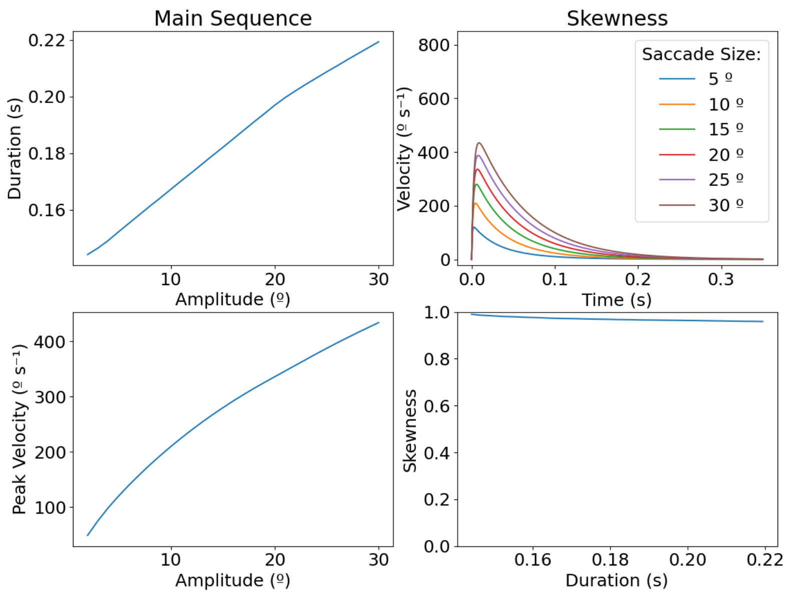


Figure 5.13: Main sequence and skewness of the saccades obtained of variant 2 with varying C

The main saccade properties still present non-linear characteristics, although there is a clear increase in the duration of the saccade as a consequence of a decrease on its peak velocity. On top of that, the velocity profiles present a decrease of their skewness with the increase of the their duration.

5.3 Variant 3

This variant is the other that has the parameter C with a pulse function of $p_m(t) = Ae^{-Bt}(1 - e^{-Ct} - Cte^{-Ct})$. This has the characteristic to perform a more realistic movement, since its velocity at the start is zero, unlike the two other variants. Just like what was done for the previous variant, this will be divided into scenarios the value of C is fixed, and the scenario C moves such that the motor velocity reaches a defined maximum velocity, approached by equation 4.31.

5.3.1 Fixed C

For a fixed C scenario, the value obtained for this parameter was obtained using the same strategy as what was done in 5.2.1: for a $30\ ^o$ saccade it is desired to have a solution with a B parameter of 35, a D parameter of 20.33 and a k_0 parameter of 17.5, which for a top velocity of $7200\ ^o\ s^{-1}$ it give $C = 3.6$. Moreover, in this situation the values obtained for the weights were: $\lambda_a = 1.5 \times 10^7$, $\lambda_e = 1$, $\lambda_d = 1.4 \times 10^5$ and $\lambda_o = 7.6 \times 10^7$. With all of this, the results obtained for the policy output were:

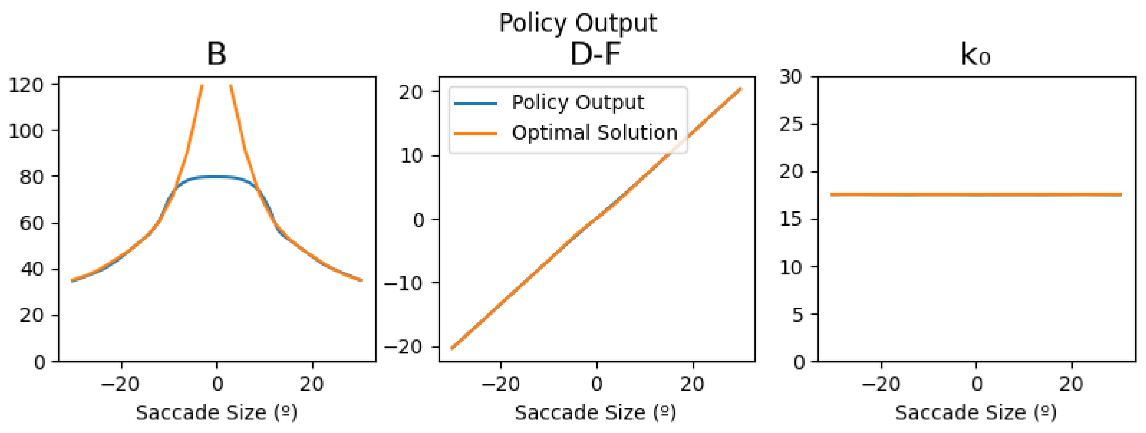


Figure 5.14: Policy output of the parameters against the desired saccade size of variant 3 with fixed C

Without the unfortunate saturation of B for lower saccades, all the parameters seem to follow the trends of the estimated values, which in turn, they are expected to produce good saccades:

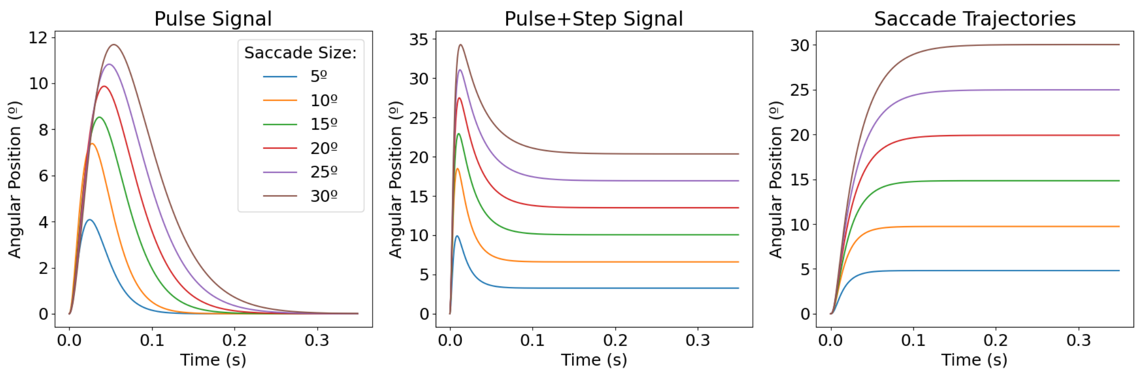


Figure 5.15: Pulse Signal, Pulse+Step Signal and Saccade trajectory output employed by the policy network of variant 3 with fixed C

Which when compared to a pure step input:

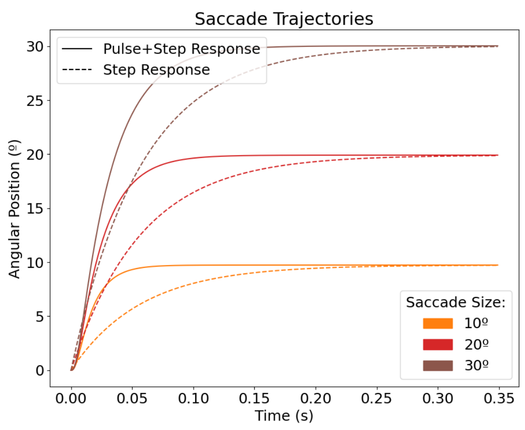


Figure 5.16: Comparison of the response between a pulse+step signal to a pure step signal for variant 3 with fixed C

That is clearly concluded- fast, efficient and no overshoot. Moreover the main sequence properties and skewness:

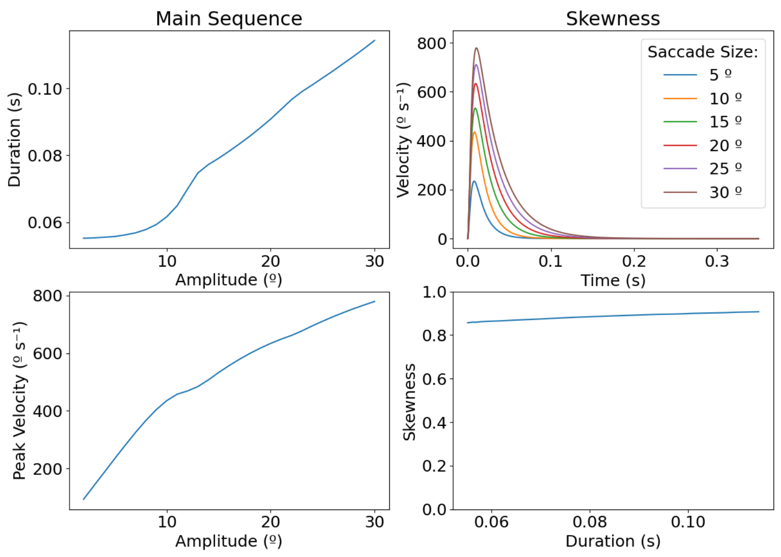


Figure 5.17: Main sequence and skewness of the saccades obtained of variant 3 with fixed C

Show all over, the characteristic of non-linear control system similar with the properties found in human data: upward non-linear relationship between peak saccade velocity and duration, peak velocity and saccade amplitude and between skewness and duration.

5.3.2 Varying C

Finally for the last case, the values obtained for the weights of the rewards were under the same conditions: $\lambda_a = 1.5 \times 10^7$, $\lambda_e = 1$, $\lambda_d = 1.5 \times 10^5$ and $\lambda_o = 7.8 \times 10^7$. With these, the values of the parameters obtained were:

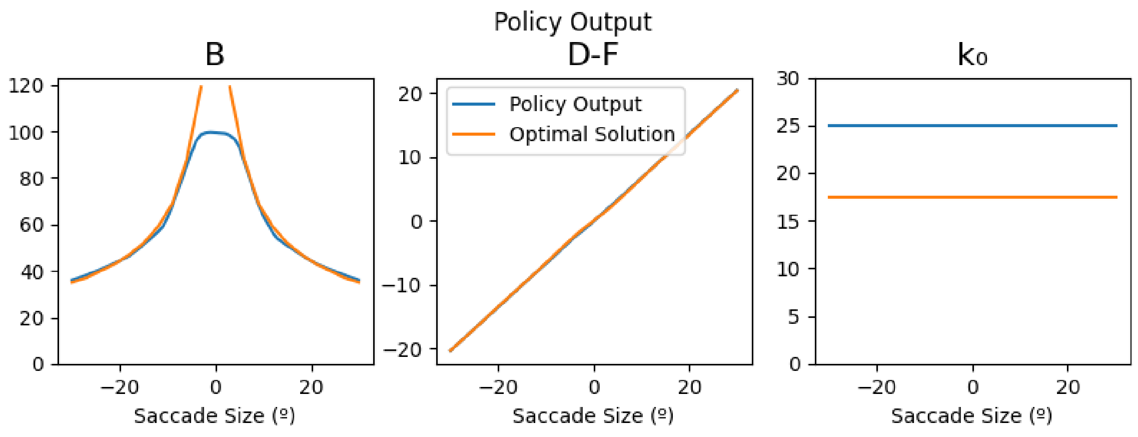


Figure 5.18: Policy output of the parameters against the desired saccade size of movable 3 with varying C

And C:

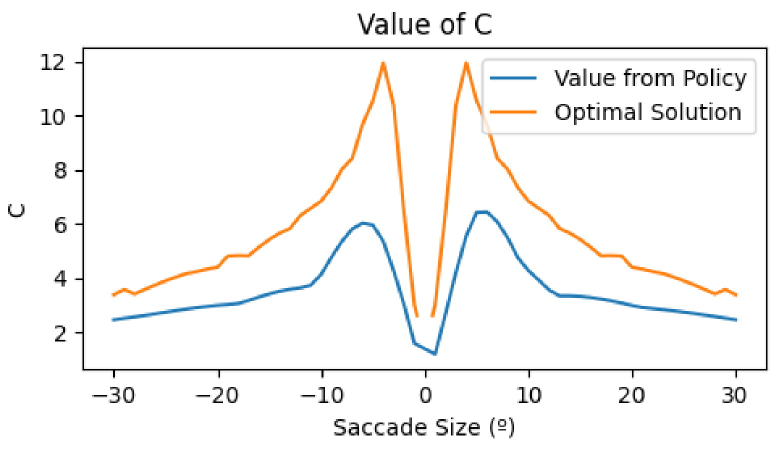


Figure 5.19: Value of C with Saccade Size for variant 3

Similarly with what was obtained for variant 2 with varying C (5.2.2), the parameter B and D follow the wanted trend, but k_0 reaches saturates to its maximum possible value. The decrease in the value of C for lower saccade sizes, is due to the saturation of B (which in this case happened to be increased mistakenly to 100); it would have kept a vertical asymptote for saccade with 0 amplitude.
Therefore, certainly, the signals and saccades produced will be similar to 5.2.2:

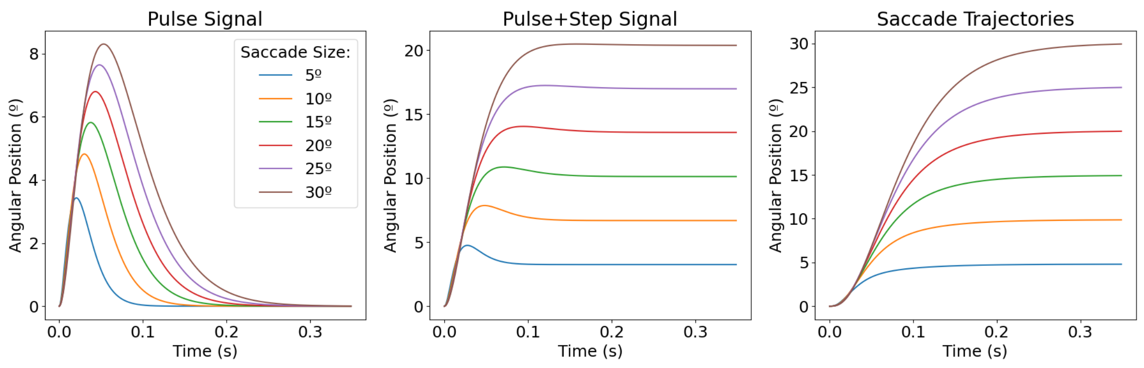


Figure 5.20: Pulse Signal, Pulse+Step Signal and Saccade trajectory output employed by the policy network of variant 3 with varying C

the pulse+step response for higher saccades, demonstrates just a step-like behaviour. And when the saccades are compared to their pure step response:

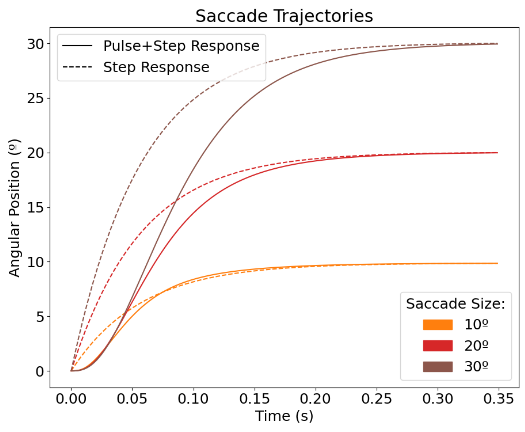


Figure 5.21: Comparison of the response between a pulse+step signal to a pure step signal for variant 3 with varying C

it produces even slower saccades due to the use of a second order step function for the raise of the pulse - so the response resembles to a second order step.

So the saccades will have the following main sequence and skewness properties:

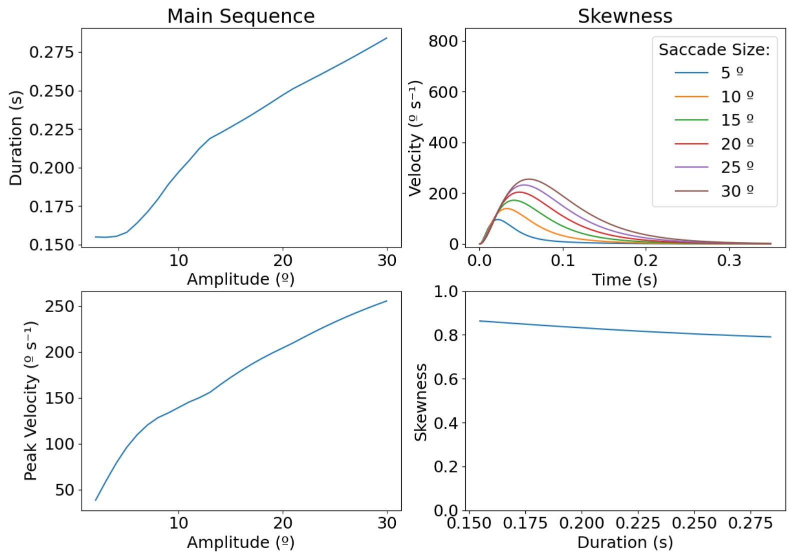


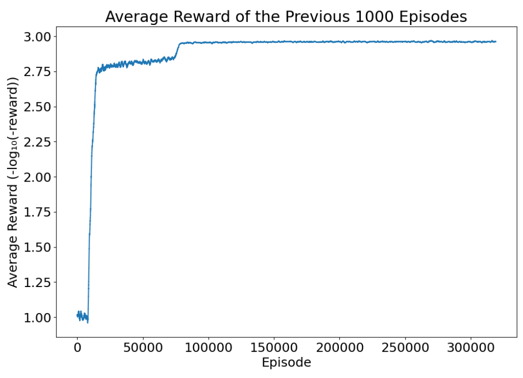
Figure 5.22: Main sequence and skewness of the saccades obtained of variant 3 with varying C

which is very similar to the graphs in 5.2.2.

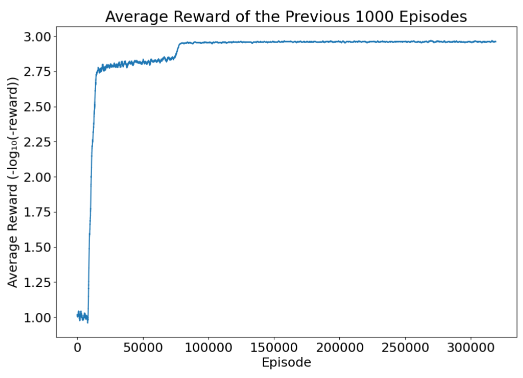
5.4 Discussion

The training done for 5.1, 5.2.1 and 5.3.1 clearly reached the outcome desired: the soft actor-critic learned the optimal across all saccade sizes - not accounting for the saturating values of B, something that doesn't invalidate the results obtained, and although it will something always present, it can improved by expanding the search region of the parameter B. They all produced faster saccades compared to a saccade from a pure step signal with a clear non-linear control behaviour solidified by the data obtained for the main sequence and skewness properties of the saccades. it can also be seen that the two less realistic variants (1 and 2) produced saccades with lower peak velocities and higher duration compared with variant 3.

Now, that's been identified that those three tests worked as intended, it would be important to understand why the other two did not. There are a few causes that could explain these problems: starting first with the training time/ number of episodes - the parameters B and D clearly reached the desired values, so maybe it could be just a matter of time for the last parameter to reach the optimal values:



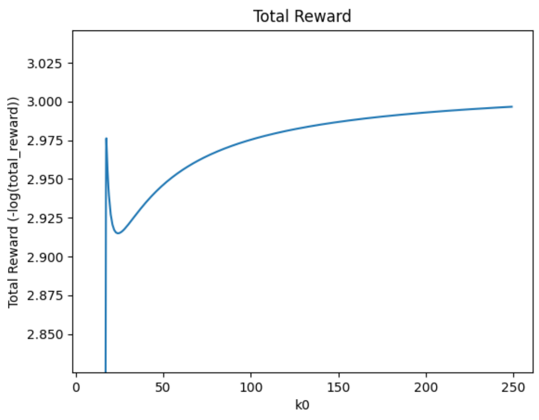
(a) Variant 2 with varying C



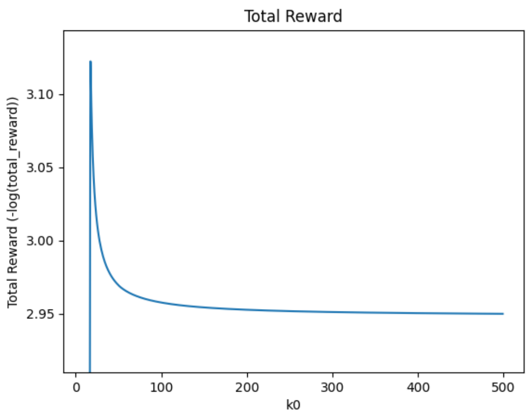
(b) Variant 3 with varying C

Figure 5.23: Average reward of the previous 1000 episodes for the 2 variants with a varying C parameter. Variant 2 is on the left and variant 3 is on the right. The scale for the reward is logarithmic one; a value of 3 would correspond to a reward of 10^{-3}.

The graph above represents the average reward of the previous 1000 episodes done obtained during the training process of the two variants with varying C. As it can be seen, for both cases, at the beginning there is a small portion of episodes that don't seem to do anything and then the average of the rewards are basically blasted upwards, which is at the point where the algorithm as enough samples to start learning. By fifty thousand episodes both of the variants reach a point where the there are no further improvements in the value of the reward until the end of the training, therefore it's not lack of training. Since the updates to the neural networks are done through stochastic gradient optimization it could be that the gradient wants to update in a direction away from the desired value - there could be a better solution than the one calculated. Therefore for one desired saccade, we want to see how the value of the total reward evolves with an increase in k_0, keeping the other parameters constant. Since the values for a $30\ ^o$ saccade are known, with a $B = 35$ and $D = 20.33$ for both variants.



(a) Variant 2 with varying C



(b) Variant 3 with varying C

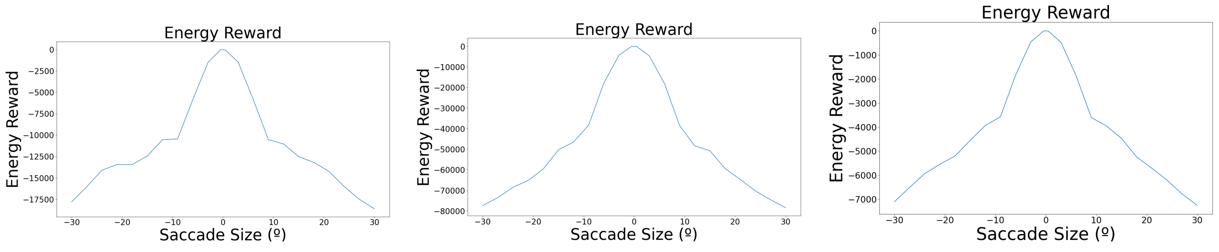
Figure 5.24: The total reward for a $30\ ^{o}$ saccade with a changing parameter k_0 and fixed parameters $B = 35$ and $D = 20.33$ for variant 2 with varying C (left) and variant 3 with varying C (right)

It can be seen in Figure 5.24 that the two variants have a maximum at a value of 17.5 for $k_0 = 17.5$, but for the case of variant 2 there are better solution beyond a value of 25. So what it means is that during learning the gradient steps is clearly leading the neural networks to those better values. This parameter influences the duration, the energy and the overshoot, but for this case influence in overshoot can be neglected because this only happens if k_0 has a value lower than the lowest pole of the yaw function ($\theta_y(s)$). So, as k_0 increases the energy utilized gets lower, but the saccades get slower, so it has longer duration, meaning that decreasing the energy spent is much better than taking shorter amounts of time under this design.

Unfortunately the same cannot be concluded for variant 3, there is a clear decrease in the reward as the value of k_0 increases after 17.5, so there is no clear justification for this behaviour in training. There must be some intricate internal behaviour that the algorithm cannot get past. It is also possible that this is present on variant 2 with varying C because it was tested for both cases a even finer restriction in the possible region of exploration of k_0, where the gradients would point undoubtedly to an optimum at 17.5, however even then, the value of k_0 not reach the desired value and would saturate.

It is not necessarily clear why it is not possible to learn using this method, but it might be because the calculation of C everytime a saccade is executed might constrain the parameters B, D and k_0. For a fixed C, its value is defined before training using the constraint of equation (4.18) for variant 2 and of equation (4.31) for variant 3, and $B = 35$, $D = 20.33$ and $k_0 = 17.5$; the value obtained for C is kept constant and used for every training episode. On the other hand, a varying C has its value calculated every episode: when the policy network outputs an action, a value for B, D and k_0, a new value of C is calculated using equations (4.18) and (4.31) for variant 2 and 3 respectively. Since C is a function of the action parameters ($C = f(B, D, k_0)$), and also a constraint from the maximum motor velocity, it is then a constraint for parameters B, D and k_0. So, instead of having the algorithm learn the optimal values of 3 free parameters, we have an algorithm trying to learn the optimal value of 3 constrained parameters. As seen from the policies obtained, this constraint hinders the learning process of the Soft Actor-Critic

algorithm. Moreover it is possible to make a parallel with the human biology and justify why the varying C method is unrealistic. In the human eye for each muscle there are thousands of motor neuron, and because each has a threshold for how much force they can use, depending on the size of the saccade, a different number of these will be recruited to exercise some force - the largest saccades would require the recruitment of all the neurons. So the constraints imposed on this design (every saccade makes the motor output the same maximum velocity) is the equivalent of forcing the human eye to use all the motor neurons when a saccade, regardless of the size, had to be executed.
Finally might be important to make a qualitative comparison between all the three variants that worked. Firstly, this comparison won't be done over the total reward obtained across all saccade sizes, simply because the values are different for all variants, nonetheless this could be done for the individual rewards: it's been identified that all saccades have desired accuracy with no overshoot, and that variant 1 has the slowest saccade duration, followed by variant 2 and then variant 3, intuitively it might be expected that in terms of energy they would have the same order from lowest amount of energy spent to highest.



(a) Variant 1 (b) Variant 2 with fixed C (c) Variant 3 with fixed C

Figure 5.25: Energy Reward as a function of the saccade size for variant 1 (left), variant 2 with a fixed C (center), and variant 3 with fixed C (right)

The graphs in Figure 5.25 were obtained by combining the output of the policy networks with equations 4.36, 4.37 and 4.38. Unexpectedly, variant 3 performs better in terms of duration and energy under these conditions. Although, there are certain nuances that were not mention that might make this comparison somewhat invalid. For variant 1, there's no way to control the maximum velocity of the motor, and for variant 3 the value of C was obtained through equation 4.31 which is actually a lower bound, therefore the condition that the maximum velocity of the motor is reached ($7200\ ^o\ s^{-1}$) for a $30\ ^o$ saccade is not met, whereas for variant 2 the maximum velocity of the motor can be controlled and met. The point here is that the parameters used to define the reward weights are equal for all the variants and were picked arbitrarily, which could mean that there might be a combination of desired parameters that could make the saccades more time and energy efficient for both variant 1 and 2.

Chapter 6

Conclusions

6.1 Achievements

In this work it has been designed a framework that uses the machine learning algorithm soft actor-critic to learn a open-loop saccadic control of a biomimetic eye. Because machine learning is a time consuming task, the model of the eye that was going to be used for training had to be simplified to speed up this process. This new simplified version turned to be a first order low-pass filter, with constants based on a system identification of the eye model it was based on (section 3.2).
Nonetheless the control that was design to be done to this model would mimic the same type of control of a human eye. This control is a pulse+step signal shaped by three parameters learned by the algorithm: B, responsible for the duration of the motor movement and further influencing the response of the eye plant, D, which leads the saccade to the desired orientation, k_0, in charge of matching with the slowest pole of the plant and helping produce saccades with a response faster than the one of a pure step signal. These last two parameters would ensure that the algorithm could train on any eye model, because they would match the plant characteristics.
The soft actor-critic learns by maximizing the rewards obtained from applying the control to the model. These rewards were calculated using metrics, defined in previous works and deemed important in various studies about the human eye: accuracy, energy, duration and overshoot of a saccade. Depending on the relative importance of each of these metrics, the optimal policy would change and defining these blindly would make the validation of the optimal solution very difficult - the algorithm could be stuck in a local optimum instead of the global one. So, a method was developed that would allow validate the trained policy: the weights were calculated from a solution selected for one specific saccade and used during training; if after training the policy for that specific saccade was the same, it could be assumed that the optimal policy had been obtained for all saccades.
The results of the training done were found to be overall successful. For three different pulse+step functions there was a match between the values for the different parameters and the estimation for the optimal solution done through exhaustive search, leading to the conclusion that the methodology used to implement this framework works. It's important to mention that 2 of those step+pulse functions had the

introduction of a another parameter, which had to be kept constant to allow the algorithm. These cases are useful because the introduction of this new parameter could be used to reflect maximum velocities that real life components would have, and confine the training done to more realistic scenarios. The saccades produced also matched the data observed in humans - the main sequence and skewness of the saccades had the same non-linear properties. Finally that the pulse+step function that obeyed a more realistic model of a motor behaviour, performed better energetically and duration wise when compared to the others, although such comparison is unfair due to the arbitrariness of the optimal solution picked for all models.

6.2 Future Work

The main framework for the application of the soft actor-critic algorithm has been setup, so what's left to do is verify its validity for the other models: applied both to the simulator version, section 3.2, and the real life version, section 3.1, as well as the version with 6 motors, each simulating one of the 6 muscles of the human eye [7], along with the expansion of the degrees of freedom - using more motors.
Other ways to explore the use of the soft actor-critic is by applying it in closed loop, which instead of defining a whole shape for the trajectory of motor command, small increments for the motor commands would be defined as the policy network output for each time step; or even follow up on the work done by Rui [6] and explore how the addition of noise and signal dependent noise to the dynamics of the model would impact the performance of the algorithm.

Bibliography

[1] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[2] U. of Illinois Chicago. Machine learning in healthcare: Examples, tips resources for implementing into your care practice. `https://healthinformatics.uic.edu/blog/machine-learning-in-healthcare/`, July 2021. Accessed: 26/10/2021.

[3] A. John, C. Aleluia, A. J. Van Opstal, and A. Bernardino. Modelling 3d saccade generation by feedforward optimal control. *PLOS Computational Biology*, 17(5):1–35, 05 2021.

[4] M. Lucas. Construction and characterization of a biomimetic robotic eye model with three degrees of rotational freedom. Master's thesis, Instituto Superior Técnico, 2017.

[5] C. Tavares. Control of saccades with model of artificial 3d biomimetic eye. Master's thesis, Instituto Superior Técnico, 2019.

[6] R. E. M. Cardoso. Feedback control of saccades on a model of a 3d biomimetic robot eye. Master's thesis, Instituto Superior Técnico, 2019.

[7] B. das Chagas e Silva Colaço Dias. Modeling, simulation, analytic linearization and optimal control of a 6 tendon-driven biomimetic eye: a tool for studying human oculomotor control. Master's thesis, Instituto Superior Técnico, 2020.

[8] How does the eye work? `https://www.optometrists.org/general-practice-optometry/guide-to-eye-health/how-does-the-eye-work/`, author = Optometrist Network, note = Accessed: 14/10/2021.

[9] M. D. Binder, N. Hirokawa, and U. Windhorst, editors. *"Retinal Slip"*, pages 3526–3526. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[10] E. Chekaluk and K. Llewellyn. Masking effects in saccadic eye movements. *Studies in Visual Information Processing*, 5:45–54, jan 1994. ISSN 0926-907X.

[11] W. E. K. R. W. Baloh, A. W. Sills and V. Honrubia. Quantitative measurement of saccade amplitude, duration, and velocity. *Neurology*, 25(11):1065–1065, 1995.

[12] J. V. Opstal. *The auditory system and human sound-localization behavior*, volume 1. Academic Press, 2016.

[13] D. A. Robinson. Models of the saccadic eye movement control system. *Biologic Cybernetics*, 1973.

[14] S. P. S. Agostino Gibaldi. The saccade main sequence revised: A fast and repeatable tool for oculomotor analysis. *Behavior Research Methods*, 2021.

[15] S.-N. Yang and G. McConkie. Eye movements during reading: a theory of saccade initiation times. *Vision Research*, 41(25):3567–3585, 2001.

[16] J. B. K. et al. *Quaternions and rotation sequences,*. Princeton University Press, 1999.

[17] M. Boyle. The integration of angular velocity. *Advances in Applied Clifford Algebras*, 2017.

[18] R. Jagtap. Understanding markov decision process (mdp). `https://towardsdatascience.com/understanding-the-markov-decision-process-mdp-8f838510f150`, September 2020. Accessed: 26/10/2021.

[19] J. TORRES.AI. The bellman equation. `https://towardsdatascience.com/the-bellman-equation-59258a0d3fa7`, June 2020. Accessed: 26/10/2021.

[20] A. Srivats. Information entropy. `https://towardsdatascience.com/information-entropy-c037a90de58f`, April 2019. Accessed: 26/10/2021.

[21] J. Lee. Probability of winning the lottery. `https://towardsdatascience.com/probability-of-winning-the-lottery-9331080952c4`, January 2020. Accessed: 26/10/2021.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL `http://arxiv.org/abs/1801.01290`.

[23] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL `http://arxiv.org/abs/1812.05905`.

[24] B. D. Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnagie Mellon University, 2010.

[25] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[26] S. Kullback. *Information Theory and Statistics*. Dover Publications, 1968.

[27] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/schulman15.html`.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

[29] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `https://proceedings.mlr.press/v48/mniha16.html`.

[30] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, 2015.

[31] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL `http://arxiv.org/abs/1802.09477`.

[32] J. Kober and J. Peters. *Policy Search for Motor Primitives in Robotics*, pages 83–117. Springer International Publishing, Cham, 2014.

[33] P. Kormushev, S. Calinon, and D. G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[34] N. Sprague and D. Ballard. Eye movements for reward maximization. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, page 1467–1474, Cambridge, MA, USA, 2003. MIT Press.

[35] Y. Liu, E. Reichle, and D.-G. Gao. Using reinforcement learning to examine dynamic attention allocation during reading. *Cognitive science*, 37, 02 2013.

[36] W. M. Samuel J. Ling, Jeff Sanny. *University Physics*, volume 1. OpenStax, 2016.

[37] Pyglet. `http://pyglet.org/`, .

[38] Development of eye movements in infants. `https://entokey.com/development-of-eye-movements-in-infants/`, author = Ento Key, note = Accessed: 17/10/2021, .

[39] K. E. Das S, Gandhi NJ. Open-loop simulations of the primate saccadic system using burst cell discharge from the superior colliculus. *Biol Cybern.*, 1995.

[40] A. T. Bahill and D. Harvey. Open-loop experiments for modeling the human eye movement system. *Systems, Man and Cybernetics, IEEE Transactions on*, 16:240 – 250, 04 1986.

[41] J. G. Ángela Molina. *Pulse Voltammetry in Physical Electrochemistry and Electroanalysis*, volume 1. Springer, 2015.

[42] U. Graf. *Applied Laplace Transforms and z-Transforms for Scientists and Engineers*, volume 1. Birkhäuser Basel, 2004.

[43] C. M. Harris. Does saccadic undershoot minimize saccadic flight-time? a monte-carlo study. *Vision Research*, 35(5):691–701, 1995. ISSN 0042-6989.

[44] R. Shadmehr, J. Orban, M. Xu-Wilson, and T.-Y. Shih. Temporal discounting of reward and the cost of time in motor control. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 30:10507–16, 08 2010.

[45] S. M.-I. Reza Shadmehr. *Biological Learning and Control: How the Brain Builds Representations, Predicts Events, and Makes Decisions*. The MIT Press, 2021.

[46] J. B. (https://math.stackexchange.com/users/1726/jim belk). Quaternion distance. Mathematics Stack Exchange. URL `https://math.stackexchange.com/q/90098`. URL:https://math.stackexchange.com/q/90098 (version: 2011-12-10).

[47] K. Ogata. *Discrete-Time Control Systems*. Prentice-Hall, Inc., USA, 1987. ISBN 0132161028.

[48] H. M. Gillen C, Weiler J. Stimulus-driven saccades are characterized by an invariant undershooting bias: no evidence for a range effect. *Exp Brain Res.*, 2013.

[49] J. Brownlee. How to choose an activation function for deep learning. `https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/`, January 2021. Accessed: 24/10/2021.

[50] S. L. Smith, P. Kindermans, and Q. V. Le. Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489, 2017. URL `http://arxiv.org/abs/1711.00489`.

[51] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[52] F. Schneider, L. Balles, and P. Hennig. Deepobs: A deep learning optimizer benchmark suite. *CoRR*, abs/1903.05499, 2019. URL `http://arxiv.org/abs/1903.05499`.

[53] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL `http://arxiv.org/abs/1609.04747`.