Pacing regulation for runners

Master Thesis

Juan Esteban Molano Valencia

Delft University of Technology

TU Delft

Pacing regulation for runners

by

Juan Esteban Molano Valencia

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday August 17, 2022 at 12:00.

Student number:	5239540	
Project duration:	December 15, 2021 - July, 2022	
Thesis committee:	Prof. dr. ir. F. A. Oliehoek,	TU Delft, supervisor
	Prof. dr. ir. M. T. J. Spaan,	TU Delft
	Dr. ir. S. Feld,	TU Delft
	Ir. R. A. N. Starre,	TU Delft
	Ir. A. Nijs,	VU Amsterdam

Cover:	Amsterdam Marathon 2021
Style:	TU Delft Report Style

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

By increasing the step frequency of the runners, it is possible to reduce the risk of injuries due to overload. Techniques like auditory pacing help the athletes to have better control over their step frequency. Nevertheless, synchronizing to a continuous external rhythm costs energy [1]. For this reason, the use of intermittent pacing may be more energy-efficient and more user-friendly for the athlete. We propose using experimental data from previous studies [2] [3], that analyzed the response of runners to intermittent pacing, to find the most efficient approach for providing the pacing. For this purpose we use reinforcement learning techniques to learn and train our target behavior. This behavior is represented as the target policy and the experimental data is assumed to be sampled using a stochastic sampling policy. However, using only a single batch of initial training data presents a problem due to the continuously increasing difference between the initial sampling policy and the target policy being learned. The use of a batch off-policy algorithm with a standard deviation correction (OPPOSD) presented in [4] is then proposed. This algorithm benefits from the advantages of the sampling efficiency characteristic of the off-policy approaches and also introduces a fixing term to tackle the mismatch between the policies. To train and evaluate the learned policies based on the algorithm, a pace behavior simulator was developed from the data of the experiments. A Markov Decision Problem (MDP) was defined on top of the simulator that determines the rules of the pacing environment that the algorithm is set to learn. After translating the experimental data into MDP-like transitions, the OPPOSD algorithm is able to learn a relatively good target policy for the pacing problem. For a future application, the resulting trained model could be deployed for real runners while still having a continuous improvement of the policy in an on-policy or off-policy approach.

Contents

List of Figures

1

Introduction

Injuries due to overload are common to find among runners of all kinds of levels. The main cause for these traumas is the inability of the lower extremity joints to effectively control the loads applied to them while training [5]. Different strategies, like the use of minimalistic footwear or changes to the footstrike patterns, have been proposed to reduce these loads [6]. A usual side outcome of applying these strategies is a beneficial increase in the athlete's step frequency. By increasing it up to 10%, the loading energy to the hip and knee joints during running can be substantially decreased [5]. Additionally, other studies show that beginner and intermediate runners tend to select step frequencies that are approximately 9% lower than their suggested optimal, defined as the step frequency at which runners consume the lowest possible amount of oxygen [7]. To achieve their optimal step frequency and also to help athletes to have better control over their cadence, techniques like auditory pacing are commonly used [8]. Runners that use this rhythmic auditory stimulation can synchronize their footfalls with a specific tempo. Nevertheless, synchronizing to a continuous external rhythm costs energy [1]. Thus the use of intermittent pacing may be more energy-efficient and more user-friendly for the athlete. To achieve the most efficient approach for providing auditory pacing it is proposed to apply machine learning techniques.

By using techniques like reinforcement learning it can be possible to train a model that is able to adapt to the athlete's preferences and increase or maintain step frequency in a user-friendly manner. However, learning techniques usually require great amounts of data for training; in this environment, the

available data is also limited. The proposed solution should be able to learn using data gathered from experiments performed with runners. Due to this limited data and the necessity of a sample efficient algorithm, a deep learning approach is considered to be the most adequate. Nevertheless, learning algorithms that use a restricted amount of data can lead to a training instability [9]. To overcome these difficulties, the most feasible approaches are the off-policy learning methods. These algorithms take into account the difference between the policy used to generate a behavior, called the *behavior policy*, and the policy that is being evaluated and improved, called the *target policy* [10] [11] [12]. This difference between policies can keep growing indefinitely and sometimes diverge when the off-policy algorithms keep using the same batch of data as input. To avoid this unbounded growing of the policies miss-match we can use a variant of the off-policy algorithms called *batch off-policy algorithms* [4]. These batch off-policy algorithms may be capable of learning a good policy using only real data from experiments with runners.

For reinforcement learning methods to learn, the model of the problem's environment must follow the Markov decision problem (MDP) or Partially Observable Markov decision problem (POMDP) principles [13][14]. For this reason, it is needed to define the pacing problem as a model with a set of states, actions, and a reward function for the agent to interact with it [13]. This model of actions and rewards will provide the algorithm with the necessary information to determine when it is necessary to activate the auditory pacing and also not to keep it on for too long.

Despite not being able to deploy and train the algorithm directly with real runners, it is still possible to estimate the performance of the resulting policy. For this reason, a simulator that emulates the runner's behavior was developed. Furthermore, this simulator allows us to create additional dummy data with similar characteristics as the data obtained from the experiments with runners. Additionally, the simulator will provide a way to evaluate and adjust our MDP model and our agents so they learn towards the most efficient approach. The resulting model should also be lightweight in terms of energy consumption and processing power to be deployed outside with real runners.

In chapter 2 the document provides a description of related work, like pacing tools previously developed, and an insight into the reinforcement learning approaches. Afterward, chapter 3 presents a description of the methods followed for building the necessary tools for the experiments. The methods section is followed by a description of the experiments performed and their results in chapter 4. Later, chapter 5 discusses the obtained results, and in chapter 6 the conclusion for the work is provided.

1.1. Research questions

In brief, the main objective of the project is to implement a Deep Reinforcement Learning algorithm solution able to learn a pacing decision policy with a relatively good performance for future use from a batch of experimental data from real runners.

Research questions 1.1.1 *Further questions that we want to answer with this project are:*

(I) *Can a trained Deep Reinforcement Learning algorithm learn a policy to provide auditory pacing efficiently?*

(II) *What is the least amount of training data necessary for the implemented Deep Reinforcement Learning algorithms to learn a relatively good pacing policy?*

(III) *Is it possible to use experimental data from real runners as training data for the implemented algorithms to learn a relatively good pacing policy?*

(IV) *How could the reached solution be deployed for real runners?*

2

Background

In this chapter all the related background information is presented. The section 2.1 explains what has been done with the purpose of regulating the step frequency for runners. The following section 2.2 gives an insight of the reinforcement learning family of algorithms and all the necessary background used for the development of algorithms in this project.

2.1. Pacing regulators

Sustaining a regular step frequency for runners is a problem that has been addressed by previous research as a way to improve the running style of athletes [15][16]. Solutions that implement mobile applications are common for regulating the pace. Applications like *PaceGuard* work as a mobile training assistant that relies on the device's sensors for calculating the current runner's step frequency and using acoustic feedback to regulate it [17]. The results obtained showed that the provided auditory pacing might help to regulate the step frequency specially to the runners with a better sense of rhythm. However, this approach was only a pilot and does not take into account any kind of energy saving since the feedback is constant during the whole run.

Another common approach for auditory pacing is by shifting the beat of the music, that the runner might be listening to, or selecting playlists with songs that match with the target tempo [8][18][19]. It was found that by using music the pacing regulation is more enjoyable for the runners, but it is not as effective when the goal is to change the cadence [8]. Furthermore, despite showing consistent

results towards increasing the cadence these approaches do not provide any measurements about the estimated energy consumption [1][20].

A proposal that implements intermittent pacing regulation provides the auditory pacing as an unintrusive regulator for the runners [21]. This application uses the device's sensors to calculate the current pace and makes use of a heuristic approach, it determines when to activate the guidance in case the runner is having difficulties for keeping the pace. This approach relies in a set of parameters (e.g. step sampling interval, pace delta tolerance, feedback message duration) that can be tuned for making the tool more effective for each runner. However, as it is remarked in the publishing, the large majority of users are not willing to configure these parameters. As a conclusion, the research finds that using a beats based auditory pacing is more effective and precise than using natural language messages. This approach can be used as a baseline that we will try to improve by the use of machine learning techniques. These techniques can help us to be more effective with the use of the pacing beats and also to avoid tuning parameters as the pace delta tolerance or the feedback message duration.

2.2. Reinforcement learning

In this section we give a summary of the most relevant concepts for reinforcement learning. Additionally, some specific relevant models are also presented.

Reinforcement learning is a part of the concept of machine learning that aims to maximize the future rewards obtained while interacting with an environment [22]. The learner retrieves information from a cause-effect principle and learns from it through a trial-and-error approach. The learning model is able to detect changes in the environment that are caused by its actions. These changes may as well be reflected as changes to the environmental states in which the agent is. In this regard, the model learns how its chosen actions affect the agent's current state and its possible future decisions.

2.2.1. Markov decision process (MDP)

For reinforcement learning models to learn, it is generally needed to model the environment as a Markov Decision Process (MDP) [13]. These kind of models are used to represent decision making problems and are usually composed of the following elements:

- **Environment**: Is an object that inherits one or multiple physical processes. The current state of the process is usually obtained by measuring the physical outcomes of the environment or by evaluating through a computerized model of the same environment.
- **State ($\mathcal{S}$)**: The state $s_t \in \mathcal{S}$ defines the state of the environment in the time step t. It changes over the time according to the environment's transition function.

- **Action ($\mathcal{A}$)**: An action $a_t \in \mathcal{A}$ is the way of interacting with the environment. Depending on the environment an action can be e.g. to turn the pacing on or off for the pace regulator problem.
- **Reward ($\mathcal{R}$)**: The reward $r_t \in \mathcal{R}$ is a scalar value that is generated by the environment. It is dependent on the transition between states s_t and s_{t+1} that was caused by an action a_t. this dependency can also be represented as the result of a reward function $r(s, a)$.

Mathematically, we consider a finite horizon MDP $M = \langle \mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma \rangle$ with a continuous state space $\mathcal{S}$ and a discrete action space $\mathcal{A}$. P is the transition probability distribution $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ is defined as the expected reward functions and $\gamma \in (0, 1]$ is the discount rate.

Additionally the interaction with the environment is done with an agent that follows a specific policy:

- **Agent** The agent is an object that interacts with the environment through actions a_t, observes the possible transition $s_t \to s_{t+1}$ and receives the reward r_t.
- **Policy** A policy $\pi(s_t, a_t)$ is the probability distribution over the set of possible actions a_t given the state s_t at a time step t. An agent is defined by its policy as each action taken is sampled using that policy.

An agent observes from the environment tuples of state, action, reward and next state: (s_t, a_t, r_t, s'). In this definition, the observed s' is the resulting next state s_{t+1}. The interaction between an agent and the environment is depicted in the figure 2.1. In here, the actor and decision maker is called the Agent, who interacts with the environment through actions a_t and receives from it rewards r_{t+1} and information about the new state s_{t+1}.

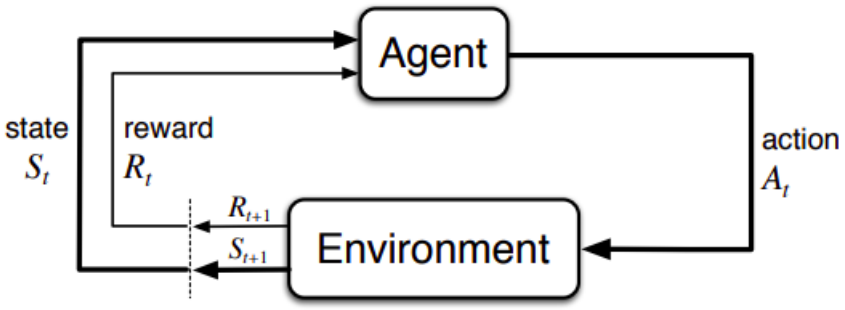

Figure 2.1: The agent-environment interaction in a MDP [22]

For the model to satisfy the Markov Property, the next state s' and the reward r must depend on the current state-action tuple (s, a). This relationship is represented by the probability function in equation 2.1.

$$p(s', r|s, a) = Pr\{r_{t+1} = r, s_{t+1} = s'|s_t = s, a_t = a\} \tag{2.1}$$

Furthermore, if the action a is sampled from the policy $\pi(s_t, a_t)$, then the Markov Property induces a conditional independence of (s_{t-1}, r_{t-1}) and (s_{t+1}, r_{t+1}) given s_t.

The agent's task is to learn a policy π that maximizes the expected future rewards. This objective is represented by the equation 2.2. This is the also known as an undiscounted finite horizon problem.

$$\arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} r_t | s_0 \right] \tag{2.2}$$

In order to bound the future reward, a discounted reward is presented in equation 2.3. Where γ is a parameter , $0 \leq \gamma \leq 1$, called the discount rate. This factor determines how future rewards are weighted in the value function from the current state. In this way, rewards that are closer to t get a higher weight than those that are occurring later.

$$\arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} \gamma^t r_t | s_0 \right] \tag{2.3}$$

T represents the time limit of the process ($T < \infty$) and p_π are the environment dynamics probabilities after following a set of actions sampled using the policy π.

2.2.2. Policy values

The main objective of reinforcement learning algorithms is to estimate the policy π that maximizes the value stated in the equation 2.3. This policy can be approximated by estimating the so called value functions. These value functions are the state value function $V_\pi(s)$ in equation 2.5 and the state-action value function $Q_\pi(s, a)$ in equation 2.6. Both of these equations use a discounted future reward R_t defined in equation 2.4.

$$R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k} \tag{2.4}$$

$$V_\pi(s) = \mathop{\mathbb{E}}_{\pi} [R_t | s_t = s] , \tag{2.5}$$

$$Q_\pi(s, a) = \mathop{\mathbb{E}}_{\pi} [R_t | s_t = s, a_t = a] \tag{2.6}$$

In general the value function can also be expressed as a function of the Q value, as shown in equation 2.7.

$$V_\pi(s) = \mathop{\mathbb{E}}_{a \sim \pi} [Q_\pi(s, a)] . \tag{2.7}$$

2.2.3. Function approximation

Usually the objective is to maximize either one, or more approximated value functions. As the expressions for $\max_\pi V_\pi(s)$ and $\max_\pi Q_\pi(s,a)$ satisfy Bellman's principle of optimality [13], they could be solved exactly using Dynamic Programming. However, for most problems this is not feasible when there are many state variables, due to the curse of dimensionality. Additionally, in reinforcement learning another issue is that usually we do not know the transition and reward functions. For instance, the value functions are usually approximated by neural networks.

Artificial Neural Networks

Neural networks are models designed for solving problems in fields as pattern recognition or data analysis [23]. The most important feature of these models is their ability to approximate the solution of a problem by using a set of training samples. They are commonly arranged in a multilayer architecture, as in figure 2.2, and each of the nodes, or processing units, usually performs a linear approximation of the input followed by a non-linear activation function (e.g tanh or $\max(x, 0)$). Neural networks take an input and by a feedforward process generates an output which is used to calculate a loss (How far is from the expected value). Then by a backpropagation process (chain rule derivative) all the values within the network are updated accordingly. Another feature from these models is their capability to generalize, which allows the Neural Network to make accurate predictions on new inputs.

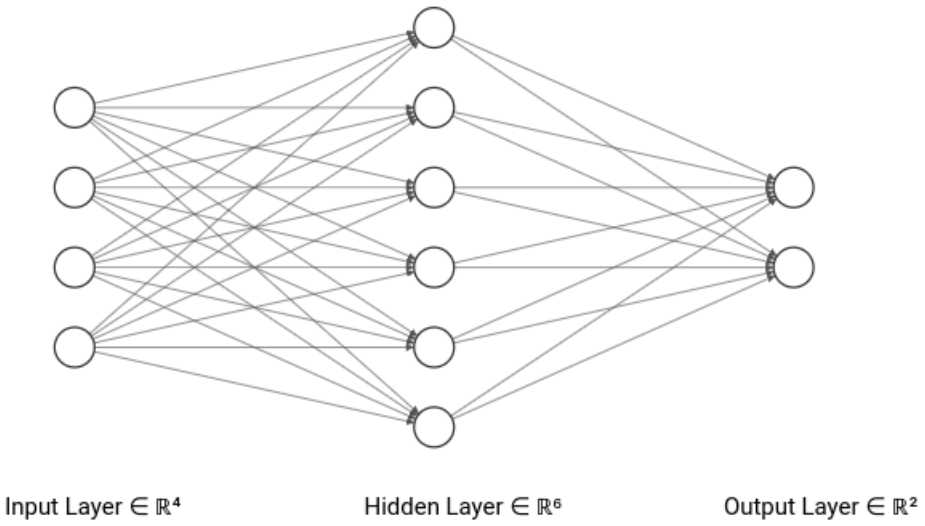

Figure 2.2: Example of the architecture of a neural network
Source: `http://alexlenail.me/NN-SVG`

A Neural Network usually receives as input a $m \times n$ matrix, where m is the input size (e.g. amount of samples) and n is the dimension of the input (e.g each of the measured variables of an input state). The

input then passes through each of the connections of a layer which are represented by many weight vectors w_i. The weight vectors have a size of $n \times h$ where n is the input dimension and h is the size of the output of the inner layer. The w_i vectors are also known as the parameters θ of a neural network. These parameters are constantly adjusted while the model learns during the training stage. The tuning of these parameters is usually done with optimization algorithms called optimizers. These optimizers (Stochastic gradient descent, Adam [24], etc.) try to find the parametric values that minimize the error when mapping inputs to outputs.

2.2.4. On- and Off-policy approaches

In general we have a *target policy* $\pi(s, a)$, which is the policy that an agent is trying to learn (can be the value function for this policy). And we will have the *behavior policy* $b(s, a)$ which is the policy that is being used by an agent for selecting an action and interacting with the environment [22].

As in reinforcement learning we do not get a training and test set to train our algorithms, all data should be sampled directly from the environment. In order to get this data and learn from it we can use the on-policy or the off-policy sampling. In the on-policy approach the behavior policy b, used for sampling, is the same target policy π which the algorithm is evaluating. This approach is relatively stable but must re-sample once the policy changes [22]. On the other side, the off-policy approach receives as input data sampled with any behavior policy b (also called sampling policy) different than the target policy π being evaluated. This allows us also to learn from older or other's experiences.

2.2.5. Exploration and exploitation trade-off

For an agent to learn a new policy, it is necessary to explore, and one way to it is by adding an extra stochastic probability $\mu = \frac{1}{\text{number of actions}}$ to the current policy π [22]. The trade-off between the stochastic actions and the current known policy is known as an ϵ-greedy approach and can be seen in the equation 2.8.

$$P(s, a) = \epsilon\mu + (1 - \epsilon)\pi(s, a) \tag{2.8}$$

This ϵ initially is set to be 1 but it should decrease with time inversely proportional to the amount of exploration done ($decisions$ is the total amount of actions taken by the agent). The hyperparameter τ is the anneal time and changes the ϵ between its maximum and minimum value, ϵ_{max} and ϵ_{min} respectively, as depicted in equation 2.9.

$$\epsilon = \max\left(\frac{1 - decisions}{\tau - 1}, 0\right) * (\epsilon_{max} - \epsilon_{min}) + \epsilon_{min} \tag{2.9}$$

2.2.6. Q-Learning approximations

Q-learning is a method that aims to estimate the policy that maximizes the state-action value [25]. This algorithm is considered a model-free reinforcement learning and tries to approximate the optimal $Q^*(s,a)$ as the addition of the current obtained reward and the future reward obtained, assuming to follow $Q^*(s,a)$ in equation 2.10.

$$Q^*(s,a) = \sum_{s'}^{T} p(s',r|s,a) \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right] \tag{2.10}$$

By interacting with the environment the algorithm gets an estimate $\hat{Q}^*$ of the current Q value (from equation 2.6) and we try to minimize the squared loss $\mathcal{L}$ between the experiences [10]. The minimizing temporal difference equation is depicted in 2.11 [22].

$$\mathcal{L}[Q] := \mathop{\mathbb{E}}_{\pi} \left[\frac{1}{2} \left(\hat{Q}^*(s,a) - Q(s,a) \right)^2 \right] \tag{2.11}$$

With Q-learning a Q online estimate from any Markov chain can be estimated iteratively using the temporal difference learning update with step size α in equation 2.12.

$$Q_{(s,a)} \leftarrow Q_{(s,a)} - \alpha \left(\nabla \mathcal{L}[Q] \right) \tag{2.12}$$

This kind of approximation is usually referred to as one step temporal difference method (TD). This solution takes time but converges to an optimal policy and it has been demonstrated in [26]. In general, the q-learning methods collect the loss over a whole episode before estimating the current Q value. One episode is defined by the history of temporal differences between the starting state $t = 0$ and an end state $t = T$.

However, in the q-learning algorithm the target policy depends on Q which is not trustworthy in the early training iterations as this is the approximating function. To have more control over the sampling of actions, also known as exploration, a sampling policy $\mu(a|s)$ is used during training. As this sampling policy $\mu(a|s)$ is different than the target policy π, the algorithm is considered an off-policy approach.

2.2.7. Policy gradient methods

Policy gradient methods aim to optimize the parameters θ of a policy $\pi_\theta(s,a)$ represented by a Neural Network.

The stationary distribution of states $d_\pi(s) \geq 0$, $\sum_s d_\pi(s) = 1$ represents a measure of how often is a state s visited [22].

Now, with the equation 2.6 as the value of the state-action pair formulation, the agent's objective

$\rho(\pi)$ is defined in equation 2.13. This objective can be interpreted as the long-term expected reward for the agent following the policy π from state s_0.

$$\rho(\pi) = \sum_s d_\pi(s) \sum_a \pi(s,a) Q_\pi(s,a) \tag{2.13}$$

Additionally, we define the performance measure ρ as the resulting value $V(s)$ from the start state of the episode s_0. Then the policy parameters are updated proportional to the gradient of ρ:

$$\nabla\theta \approx \alpha \frac{\partial \rho}{\partial \theta} \tag{2.14}$$

,

where α is a positive-definite step size.

Then,

$$\frac{\partial \rho}{\partial \theta} = \mathcal{L}_\pi[\theta] = \sum_s d_\pi(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta} Q_\pi(s,a) \tag{2.15}$$

is the policy gradient on which gradient ascent on the policy can be performed in order to maximize ρ.

A proof and a thorough discussion of this derivation can be found in [27].

Now, by using $\frac{1}{\pi(s,a)} \frac{\partial \pi(s,a)}{\partial \theta} = \frac{\partial \ln \pi(s,a)}{\partial \theta}$, assuming $Q_\pi(s,a) \approx \sum \gamma^t R_t$ and with the definition of the expected value of a function $\mathbb{E}_{x \sim p(x)}[f(x)] = \sum_x p(x) f(x)$, the equation 2.15 yields to equation 2.16. Which gives an approximate function for the loss $\mathcal{L}_\pi$.

$$\mathcal{L}_\pi[\theta] := - \mathop{\mathbb{E}}_{\pi_\theta} \left[\sum_{t=0}^{T} \gamma^t R_t \nabla \ln \pi_\theta(a_t|s_t) \right] \tag{2.16}$$

In order to find a function approximation using policy iteration, we adjust the value of the parameters θ with the formula

$$\theta_{k+1} = \theta_k + \alpha_k \mathcal{L}_\pi[\theta] \tag{2.17}$$

where α is the learning step. This is the policy gradient algorithm and we will refer to it as the basic reinforce learning algorithm (RL) [22].

Reinforcement with baseline algorithm

As the policy parameters θ are updated using random samples, it can result in a high variability of the cumulative reward values $R_t = \sum_{t=0}^{n-1} \gamma^i r_{t+i}$. This happens because each of the sampled trajectories can deviate indefinitely from each other [22].

One way to reduce this variance is to subtract the cumulative reward R_t with a baseline which is usually the approximated value $V_{\pi_\theta}(s_t)$. This is considered to be a bias free estimate that does not induce a change in the gradient [22].

Additionally, in practice the state and action spaces are large, thus the expectations with respect to s and a are approximated using $\mathbb{E}_{x\sim p(x)}[f(x)] = \frac{1}{n}\sum_n f(x), n \to \infty, x \sim p(x)$. These expectations are sample based and rely on using enough samples of s and a in order to obtain a reasonable approximation. For instance, using these approximations and subtracting the $V^{\pi_\theta}(s_t)$ term to equation 2.16, it yields

$$\mathcal{L}_\pi[\theta] = -\frac{1}{N}\sum_{t=0}^{N-1} \gamma^t \ln \pi_\theta(a_t|s_t)(R_t - V_{\pi_\theta}(s_t)) \tag{2.18}$$

Here N is the number of samples drawn which is in practice usually $N = 1$ for online updates.

This new loss $\mathcal{L}_\pi[\theta]$, defined in equation 2.18, can be replaced in the equation 2.17 to update the reinforcement algorithm.

Actor-critic algorithms

The Actor Critic methods are a group of RL algorithms where there are two functions (actor and critic) that are updated in a turn based fashion [22]. Intuitively the critic evaluates the action taken by the actor that uses this evaluation to scale its gradient update.

The estimated $V_{\pi_\theta}(s_t)$ in equation 2.18 sets a baseline for the final return, nevertheless this is independent from the transition's action and for this reason cannot be used to evaluate that action. In the actor–critic methods, on the other hand, the state-value function is applied also to the second state of the transition $V_{\pi_\theta}(s_{t+1})$ [22].

This technique, commonly known as bootstrapping, replaces the return R_{t+1} with the approximated future value $V_{\pi_\theta}(s_{t+1}) = \mathbb{E}_\pi[R_{t+1}|s_{t+1}]$. The approximation $A_{\pi_\theta}(s_t, a_t) = r_t + \gamma V_{\pi_\theta}(s_{t+1}) - V_{\pi_\theta}(s_t)$ is known as the advantage [28].

With the advantage A_{π_θ} as critic and the policy π_θ as actor, the equation 2.19 represents the loss $\mathcal{L}_\pi[\theta]$ used for most of the actor-critic algorithms.

$$\mathcal{L}_\pi[\theta] = -\frac{1}{n}\sum_{t=0}^{n-1} \gamma^t \ln \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t) \tag{2.19}$$

2.2.8. Off-policy gradient methods

Off-policy methods encounter the issue of the distributional mismatch between the target policy π and the behavior policy b described in 2.2.4 [22]. Additionally, after t steps with the policy π we encounter

a new state distribution $d_\pi(s)$. In order to adjust the distributional mismatch, the importance sampling (IS) term $\frac{d_\pi(s_t)}{d_b(s_t)}$ is added. For instance, using again $\mathbb{E}_{x\sim p(x)}[f(x)] = \frac{1}{n}\sum_n f(x), n \to \infty, x \sim p(x)$ and the IS, the gradient equation 2.15 for off-policy updates should be rewritten as

$$\nabla_\theta \mathcal{L}_\pi[\theta] = -\nabla_\pi \mathop{\mathbb{E}}_{b}\left[\sum_{t=0}^{n-1} \frac{d_\pi(s_t)}{d_b(s_t)} \gamma^t Q_\pi(s_t, a_t) \frac{\pi_\theta(a_t|s_t)}{b(a_t|s_t)}\right] \tag{2.20}$$

The off-policy gradients requires the Q-value function Q_π (or value V_π) of π_θ. In some implementations the IS term is ignored. There are many solutions to overcome this distributional mismatch, some use importance sampling approximation or other use variance reduction techniques [29] [30].

Off-policy Actor-critic

One popular implementation for off-policy gradient algorithms is the off-policy actor critic (Off-PAC) [11]. This method takes advantage of the temporal-difference technique enabling a target policy π to be learned while following and obtaining data from another policy b. The Off-PAC is intended to learn in an online and incremental form. This means, that the behavior policy b is continuously interacting with the environment and is also constantly being updated with the most recent target policy π after a certain amount of off-policy iterations C. The off-PAC approximation of equation 2.20 using the advantage term $A^{\pi_\theta}(s_t, a_t)$ is depicted in the equation 2.21.

$$\nabla_\theta \mathcal{L}_\pi[\theta] \approx -\nabla_\pi \mathop{\mathbb{E}}_{b}\left[\sum_{t=0}^{n-1} \gamma^t A^{\pi_\theta}(s_t, a_t) \frac{\pi_\theta(a_t|s_t)}{b(a_t|s_t)}\right] \tag{2.21}$$

However, this kind of algorithm due to the dismissal of the IS term tends to be slightly unstable [31].

Proximal Policy Optimization

A way to diminish the instability of the Off-PAC is proposed in [32]. This approach, called proximal policy optimization (PPO), keeps the learning policy π_θ in a trust region around the old policy $b = \pi'_\theta$. This region approximation is implemented by clipping the policy ratio outside $1 \pm \epsilon$ which means that only updates the policy for values where the loss improves and for the rest updates using the clipped value. In the figure 2.3 are shown the clipping boundaries for the policy update.

In equation 2.22 is defined the trust region for the policies, where DKL is the Kullback Leibler Divergence. The DKL is a measure of how much a probability distribution $b(\cdot|s_t)$ differs from a second probability distribution $\pi_\theta(\cdot|s_t)$ [33].

$$\min_\theta \mathcal{L}_b[\theta] \;\text{ s.t. }\; \mathop{\mathbb{E}}_{b}\left[\sum_{t=0}^{n-1} D_{KL}[b(\cdot|s_t) \parallel \pi_\theta(\cdot|s_t)]\right] \tag{2.22}$$

Nevertheless, The PPO method heavily relies on the proximity between the target and the behavior

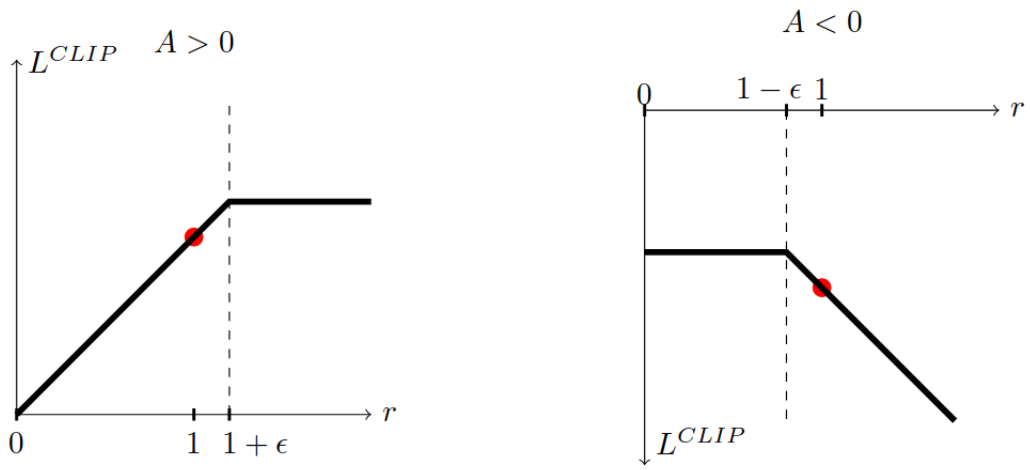

Figure 2.3: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio $r = \pi/b$, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Image from [32]

policies to work. For instance, the use of this approach, even though it leads to faster convergence, needs to iteratively update both policies in an online fashion.

2.2.9. Batch Off-policy approach

For some problems, it is assumed that a batch of data has been previously collected using a known behavior policy b. If a RL algorithm were to be trained using only this batch, we called it a batch off-policy approach. For these kind of problems the mismatch between the behavior and the target policies may grow indefinitely, and even become unbounded given its long-horizon nature. In these kind of situations, IS estimators suffer from an excessively high variance [29]. For instance, the algorithm Off-PAC in 2.2.8, that ignores the IS, may have a good performance but could not adapt correctly when the distributions mismatch becomes uncertain [4].

Off-policy Policy Gradient with State Distribution Correction (OPPOSD)

In order to avoid the exploding variance faced by existing methods, in [4] is proposed an off-policy estimator that applies IS directly on the stationary state distribution $d_\pi(s)$. The main difference is the state distribution IS $\frac{d^\pi(s)}{d^b(s)}$, which is estimated using samples collected from b.

Using $\rho_\pi(s,a) = \frac{\pi(s,a)}{b(s,a)}$, the equation 2.23 is defined.

$$\Delta(w_{\theta_w}; s, a, s') := w_{\theta_w}(s)\rho_\pi(s,a) - w_{\theta_w}(s'). \tag{2.23}$$

In equation (2.23), the term w_{θ_w} is a neural network parameterized by θ_w and represents the IS function approximation.

This function $w_{\theta_w}(s)$ is then plugged into 2.20 and we get the equation

$$\nabla \mathcal{L}_\pi[\theta] = -\nabla_\pi \mathop{\mathbb{E}}_{b} \left[\sum_{t=0}^{n-1} w_{\theta_w}(s) \gamma^t Q^\pi(s_t, a_t) \rho(s, a) \right] \tag{2.24}$$

.

This loss function $\nabla \mathcal{L}_\pi[\theta]$ is used to update the target policy π in an off-policy way. This means that the target policy π learns using only a given initial buffer of transitions sampled from b. Additionally, as this method theoretically corrects the distributional mismatch, it is no longer necessary to perform the online policy updates $b \leftarrow \pi$.

This method provides the desired behavior we want to implement for this pacing regulation project, being the experimental data the initial buffer of transitions sampled from b.

3

Methods

In order to answer the questions in 1.1, a set of activities for the project is defined.

- Develop a pace simulator
- Implement RL algorithms
- Integrate agents and simulator
- Perform experiments

A flowchart with the main tasks for each of the activities is shown in the figure 3.1. Blue represents the activities related with the deep learning algorithms. These are the tasks of creating the proper framework defined in appendix A and adding to it all relevant algorithms for the project. Purple represents the tasks related to the runner's pace simulator. Here a statistic analysis of the provided data is performed in order to develop a proper mathematical model. Afterwards, green represents the tasks related to the MDP pacing environment that should be created using the values from the simulator for it to later be used with the agents. Light orange represent the tasks related to the integration between the RL agents and the defined MDP. In here the reward model is adjusted as well as the agent for them to be able to work together. Also, the definition of the behavior policies and the samples from which the off-policy agents are expected to learn. Finally, dark orange represents the experiments performed using the integration between the agents and the simulator. In here we will test if the agents are able to learn using limited data and their performance in the simulator will be tested as well.

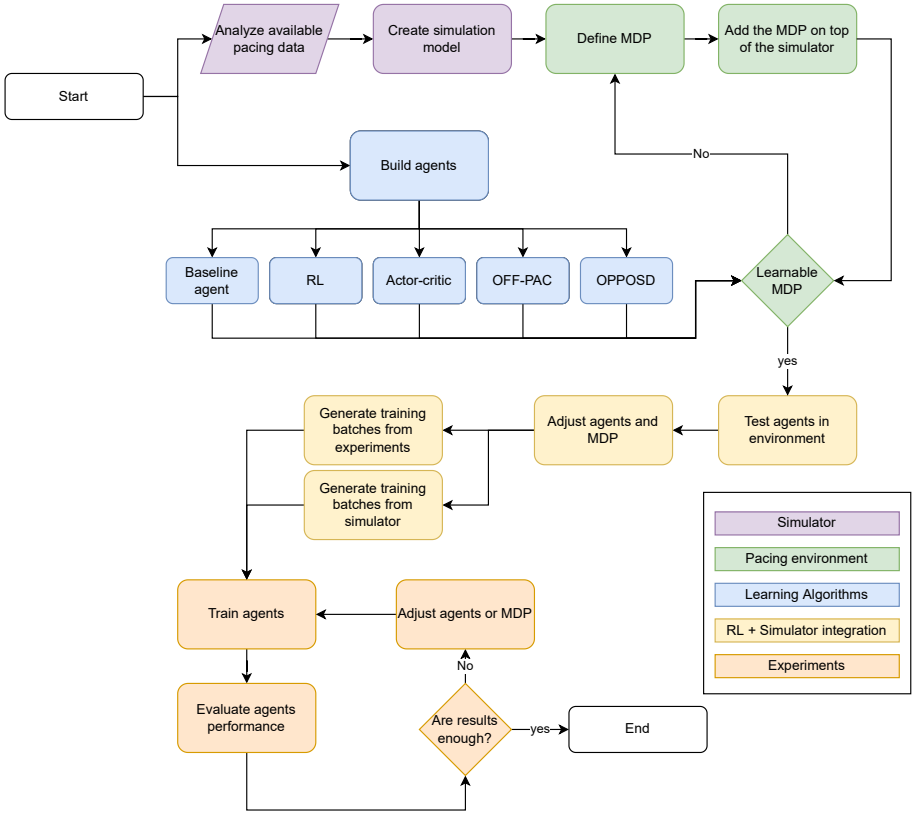

Figure 3.1: Planned activities flowchart

3.1. Provided data

A subset of data gathered in previous research [2] [3] was used. The experiments were conducted on an instrumented treadmill. After some time running, when the athlete confirms to feel comfortable with the current speed and pace, these are recorded as their preferred speed and pace. Afterwards, taking into account the preferred speed the subsequent experiments were performed. These datasets were modified specifically for this project and consist of 8 types of trials.

1. First trial of participants (16) running without pacing, for the purpose of calculating their preferred pace.
2. A second trial of participants (16) running without pacing (similar to the first one).
3. Trial in which the participants (16) ran with pacing at a frequency 10% faster than their preferred step frequency, followed by a period without pacing.
4. Trial in which the participants (16) ran with pacing at a frequency equal to their preferred step frequency, followed by a period without pacing.
5. Trial in which the participants (9) ran with continuous pacing (CP) at a frequency 10% faster than their preferred step frequency.

6. Trial in which the participants (9) ran with intermittent pacing (IP) at a frequency 10% faster than their preferred step frequency.

The raw data came from the treadmill, which was instrumented with kistler force plates. This provided data consisted of:

- The mediolateral, anteroposterior and vertical forces (N).
- The centers of pressure in x and y directions (m).
- Indicator value to measure the footstrike moment (1 or 0).
- Indicator for showing the moment of the auditory pacing activation (1 or 0).

From this dualbelt raw data the calculated variables for the pacing analysis are the following:

- Footstrikes interval (s)
- Beeps interval (s)
- Step frequency ($steps/min$)
- Pacing frequency ($beeps/min$)

3.2. Simulator

As our model will not be able to learn and adapt directly from a real environment, it is required to develop a way to test our resulting policies. For this reason a simulator that can emulate the response of the pace from runners to an external auditory pacing was developed. In order to simulate the runners response to the auditory pacing, it was necessary first to analyze the available data. From the results of the analysis of the available data, a mathematical model can be derived in order to get the desired behavior that we want to replicate.

3.2.1. Analysis of the provided data

To begin analyzing the provided data the first step was to visualize it. This way we can begin to identify general trends in the behavior of the runners while being paced and when not. Additionally, in order to be able to compare different runners with different characteristics, the pacing values are normalized. This normalization is done using the *pacing frequency* (beeps frequency which is the target and does not change) as reference. Then the observed values are $\frac{stepfrequency}{pacingfrequency}$. In the figure 3.2 examples of the pacing trend for some runners are shown.

In figure 3.2 we can observe with blue crosses (x) the paced steps of the runner and with orange crosses the non-paced steps. The time steps are the actual steps of the runner and are relative to the

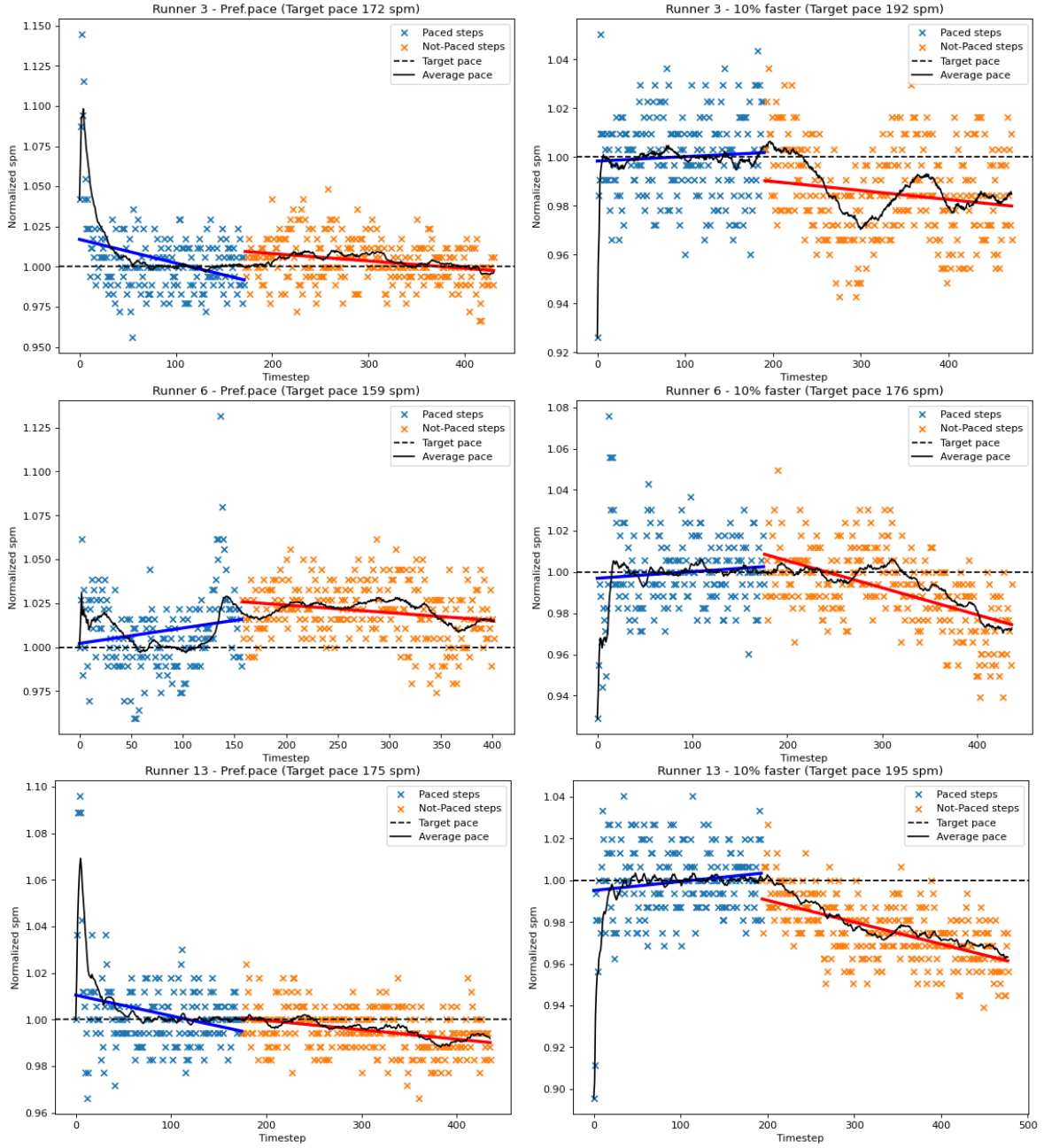

Figure 3.2: Visualized normalized pace of runners 3, 6 and 13 from the trials 1 and 2.

current cadence. So if the measured step frequency of a runner for a given time step is of $180 steps/min$ then the depicted time step will be equivalent to $\frac{60s*1step}{180steps/min} \approx 0.33s$. The objective for the runner is to try to keep a pace as close as possible to the target pace marked by the black dashed line. To have a better idea of where the real pace of the runner is, an exponential weighted moving average (EWMA) is represented as the black solid line. Finally, the trends for the paced and non-paced steps are depicted by the blue and red lines respectively.

3.2.2. General trends of the data

These graphs helped to identify 4 different pacing situations among the runners:

1. Activate auditory pacing at their preferred pace. This was done to help the runners to synchronize and help them to stick to the pace for some time before turning the pacing off.
2. Activate auditory pacing at a higher pace than the preferred one.
3. When stopping the auditory pacing at the preferred pace to test the runners ability to keep their pacing.
4. When stopping the auditory pacing after pacing at a higher pace than the preferred one.

For each of these cases separated we calculated the pacing moving average and these values were averaged for all the runners in the provided data from the trials 1, 2, 3, 4, 6 and 8. The resulting general average behavior per time step with its standard error in light blue can be observed in the figure 3.3. Here it can be seen that, for the pacing behaviors 1 and 2, there is a clear trend towards the target pace (1). This behavior indicates that in general the runners are able to synchronize their cadence using auditory pacing. These general behaviors could be modeled by a logarithmic function. On the other side, for the behavior 3 and 4 the trend downwards is clear. This suggests that runners after being paced at higher cadences tend to decrease their step frequency once they do not receive the auditory pacing anymore.

3.2.3. Modeling the trends

From the 4 different general behaviors found in 3.2.2 a regression could tell us a mathematical model to represent these trends. For behaviors 1 and 2 a logarithmic regression was done and the equations 3.1 and 3.2 model these situations. For the behavior 3 and 4 a linear regression presented in the equations 3.3 and 3.4. In these presented equations the x is the step, and y is the normalized spm.

$$y = -8.9 \times 10^{-3} \log x + 1.0396 \tag{3.1}$$

$$y = 2.614 \times 10^{-3} \log x + 0.9863 \tag{3.2}$$

$$y = -3.1926 \times 10^{-6} x + 1.0123 \tag{3.3}$$

$$y = -3.1703 \times 10^{-5} x + 0.986 \tag{3.4}$$

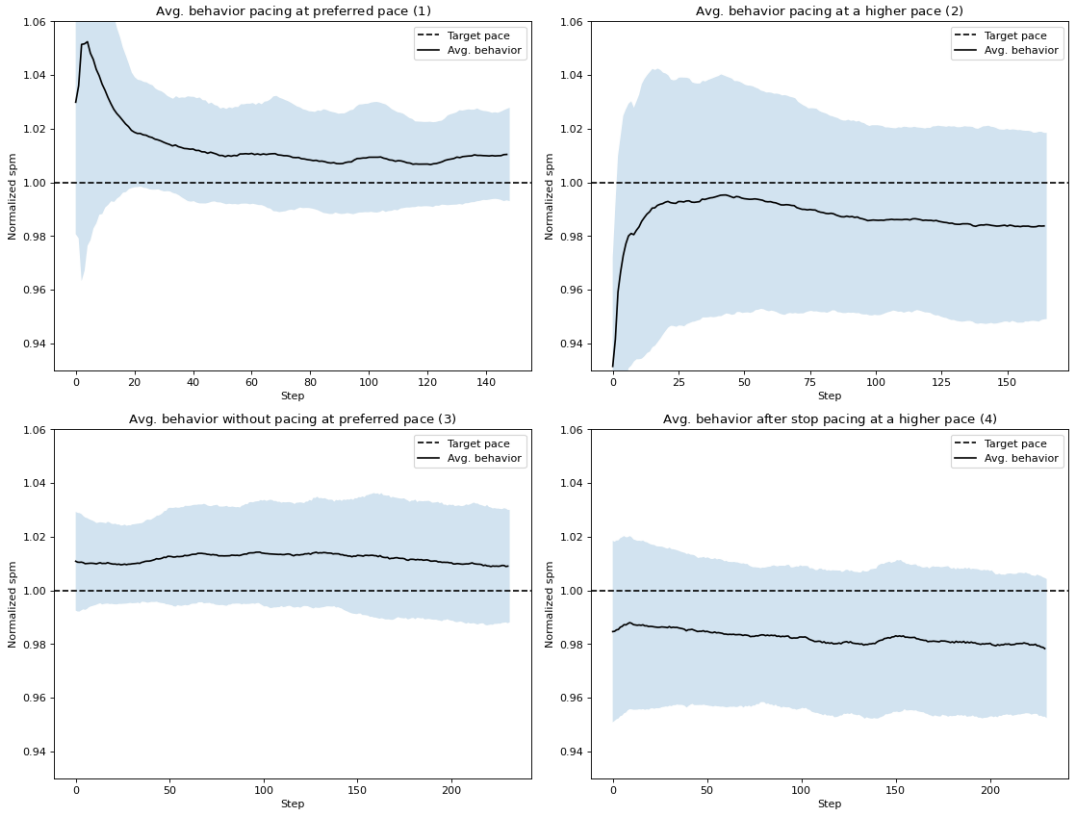

Figure 3.3: Average pacing behaviors for the 4 identified cases in 3.2.2

The closeness of these equations can be better appreciated in the figure 3.4. Given that in the experimental data the time that the auditory pacing was activated was generally shorter than the time when it was off, the x-axis is shorter for the behaviors 1 and 2 compared to the behaviors 3 and 4 in the figure 3.4.

3.2.4. Adding randomness to the models

However, if we want to emulate different runner's behaviors an additional random component should be added to the formulas without changing the general trend. With this component we can guarantee that not all the simulated behaviors are going to be the same but they still follow the desired trend.

$$y = -8.9 \times 10^{-3} \log \epsilon x + 1.0396 \quad \epsilon \in [0.2, 1.5] \tag{3.5}$$

$$y = 2.614 \times 10^{-3} \log \epsilon x + 0.9863 \quad \epsilon \in [0.2, 1.5] \tag{3.6}$$

$$y = (-3.1926 \times 10^{-6} + 1 \times 10^{-3} \epsilon) x + 1.0123 \quad \epsilon \in [-3, 3] \tag{3.7}$$

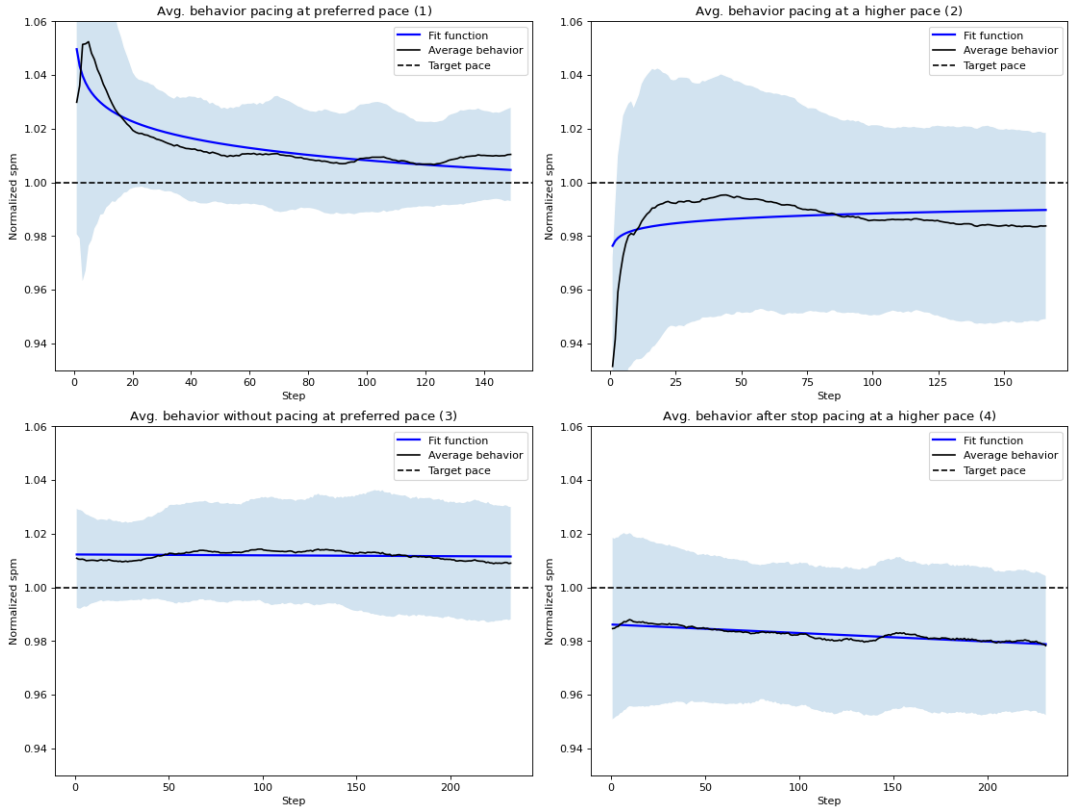

Figure 3.4: Modeled average behaviors from the experimental data

$$y = (-3.1703 \times 10^{-5} + 1 \times 10^{-3}\epsilon)x + 0.986 \quad \epsilon \in [-3, 3] \tag{3.8}$$

Equations 3.5, 3.6, 3.7 and 3.8 represent the models for the behaviors 1, 2, 3 and 4 respectively. In these a random value ϵ is added to induce the desired differences. The intervals for these values were chosen by hand searching for numbers that do not end up diverging outside of the error area from the average behavior in the figure 3.3. In figure 3.5 are shown the general behaviors and 3 different generated functions for each of them.

3.2.5. Transition between models

The next task for the simulator is to make the transition between behaviors smooth. This means that when the pacing is activated and we are following one model, as soon as we stop pacing and another model is running, we will not have big gaps between the transitioning pacing values. For this purpose, it is needed to keep track of the time step value and the last value obtained from the model.

For the logarithmic model the inverse function is then calculated using the equation $x = e^{\frac{y-z_0}{z_1}}\epsilon^{-1}$. In this equation z_0 and z_1 are the respective coefficients in the models from 3.5 and 3.6 when having the form $y = z_0 \log x + z_1$. This function takes the last value of normalized steps per minute calculated

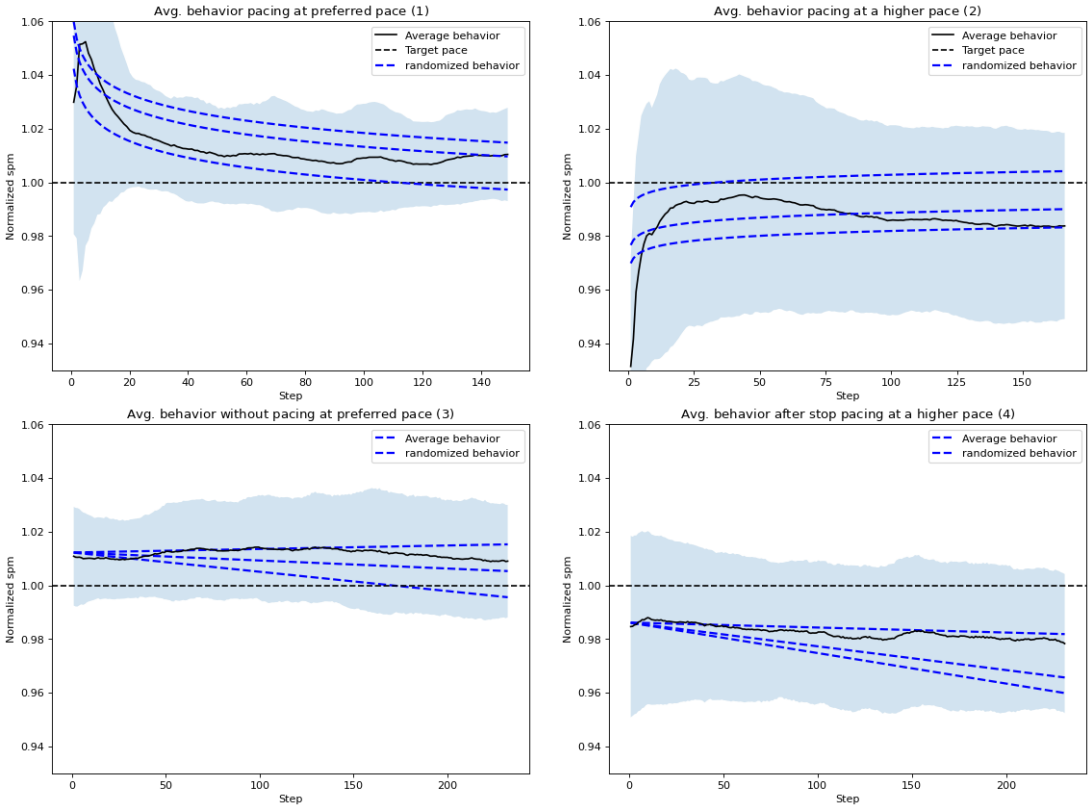

Figure 3.5: Modeled random behaviors

(y) and returns the initial x (time step) from which we need to count the time when transitioning to a logarithmic function.

For the linear functions we just adjust the intercept b in the function $y = mx + b$. Where y is the model output in normalized spm, x is the current time step, m is the slope and b is the intercept with the y axis.

When transitioning between models we expect a behavior similar to the example shown in figure 3.6. In here we start with a linear model (behavior 4) for 100 steps. Afterwards we transition to a logarithmic model (behavior 2) for other 100 steps and this one approaches us to the target pace 1. A final transition to a second linear model (behavior 4) is then performed for another 100 steps.

Each time that we make a transition all the random components are recalculated again. This way we can be certain that each simulated scenario is different.

3.2.6. Simulator results

After having the average pace behavior that we want to simulate, it is now necessary to make it look similar to the original data. For this purpose two different noises are added to the models. One noise, $n_{sim} \in [0, 5 \times 10^{-3}]$, is added directly to the simulated output y. The other noise, $n_{pace} \in$

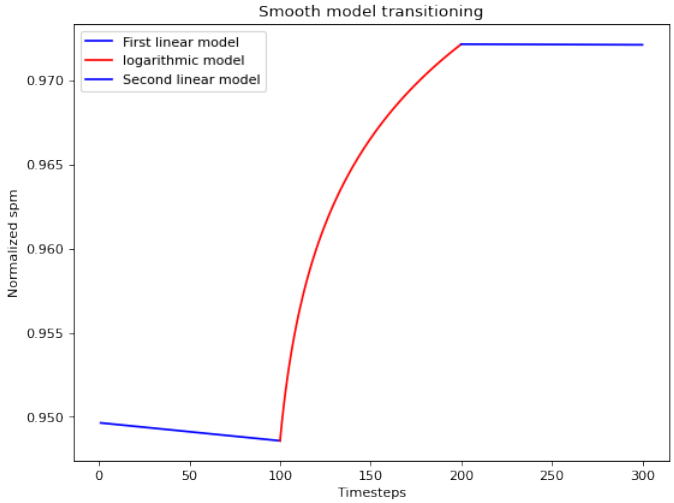

Figure 3.6: Model transition example

$[pace_{target} - 4, pace_{target} + 3]$, is added in order to denormalize the output and convert it into spm. The denormalization is done in order to get an output from the simulator which resembles the experimental data in behavior and magnitude. The added values for the noise were chosen by hand in order to approximate the real data standard deviation from the experiments. The final pace simulated value is calculated in the equation 3.9. The $pace_{target}$ is a value set by the runner and y is the output of the corresponding model.

$$pace_{sim} = (y + n_{sim}) * n_{pace} \tag{3.9}$$

To begin with the simulation the program selects a random initial pace ($pace_{init} \in [0.9, 1.08]$). From this value onwards depending on the input, if pacing is activated or not, a new value is calculated. The simulator has a state-full behavior, given that it keeps an internal track of the current model being used and the time step.

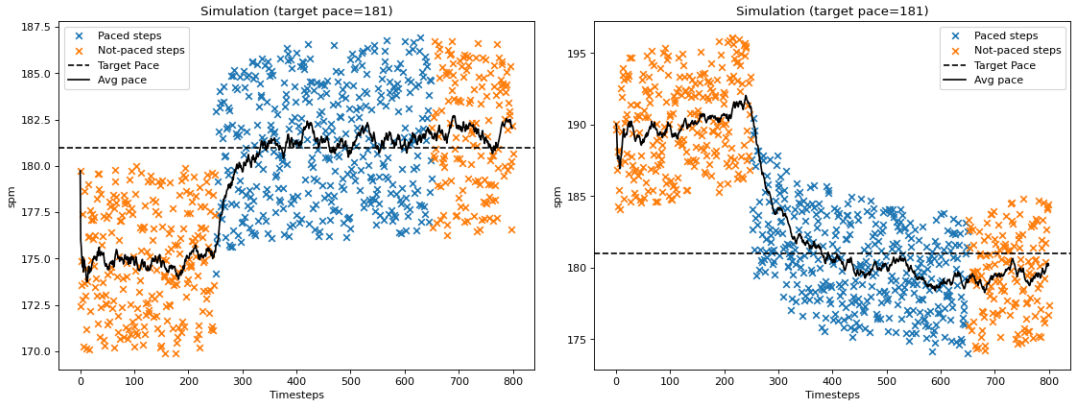

Figure 3.7: Two different simulated scenarios using the same target pace (181).

In figure 3.7, two different simulated scenarios, using the same target pace for the model, can be seen. Both of them had 250 time steps of not pacing followed by 400 paced steps and then turning off the pacing again for 150 steps.

In figure 3.8 the similitude between the provided data and the simulated one can be appreciated. On the left side images are three different behaviors from three different runners. On the right side are simulated behaviors starting on a similar pace, then activating the pacing for a similar amount of time and then stopping it. The target pace is the same for both sides.

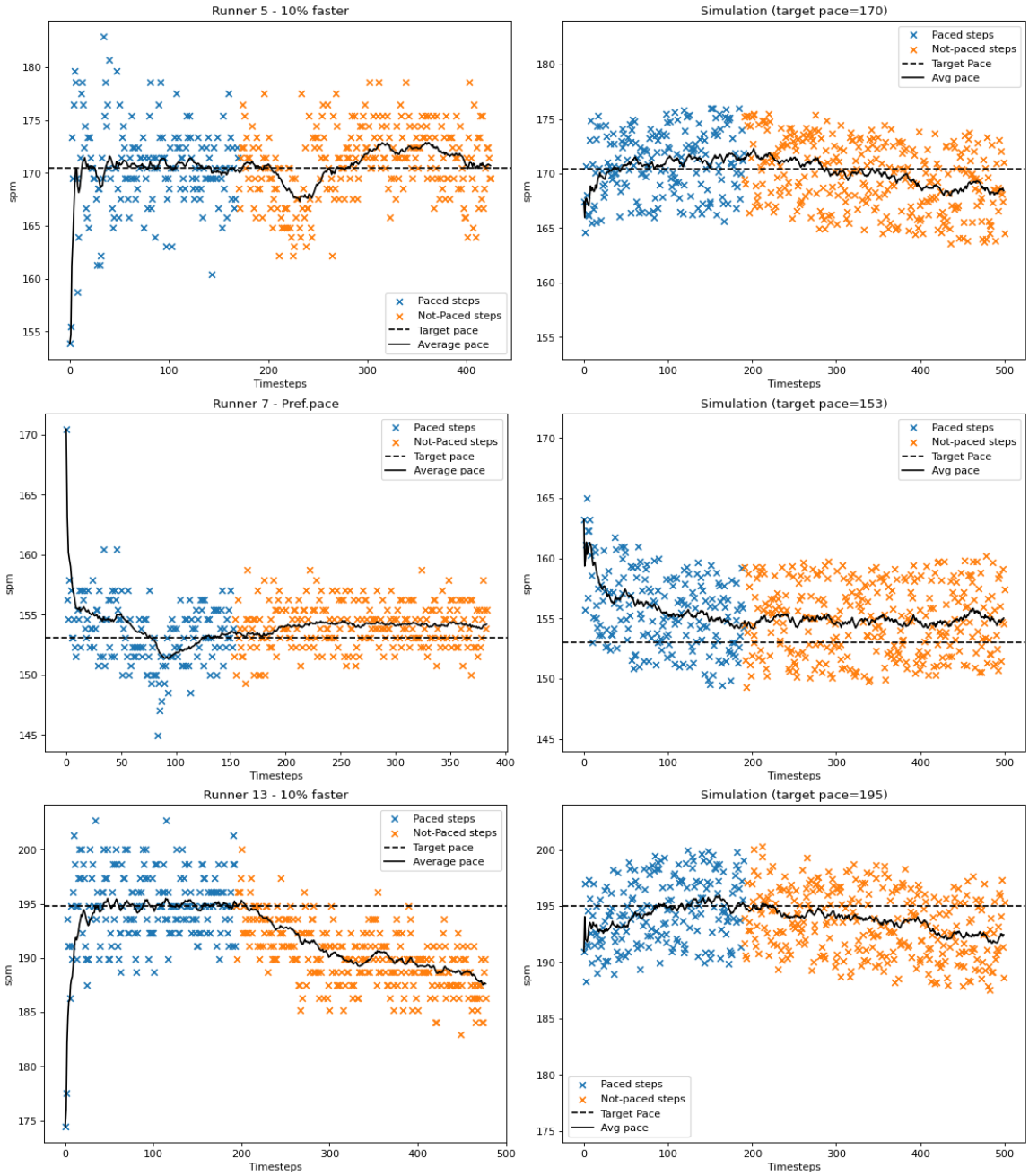

Figure 3.8: Comparison between provided data and simulated results

3.3. Pacing environment

In order to use the simulator in a way in which reinforcement learning algorithms could interact with it is necessary to define an MDP on top of it. The MDP can help us to tell the simulator for how long to pace or not to pace. After, the environment will return to us the new state in which we are, the respective reward for our action and whether the environment is done or not.

3.3.1. MDP definition

First of all, the set of states was defined as the normalized average pace at a specific timestep $\Delta pace_t = \left|\frac{\overline{pace} - pace_{target}}{pace_{target}}\right|$, where $pace_{target}$ is the target pace and $\overline{pace}$ is the Exponentially Weighted Moving Average (EWMA). The EWMA is calculated using the formula $\overline{pace}_t = \alpha pace_{sim} + (1-\alpha)\overline{pace}_{t-1}$ where $pace_{sim}$ is the measured pace in the time step t and α is a weight value chosen to be by default as 0.95.

$$S_t : \{\Delta pace_t\}$$

In here, we define blocks of steps per action. This is done given that in real life it does not make sense to pace for less than 10 seconds, otherwise the runner would not have enough time to synchronize with the auditory pacing. For this reason, the minimum block is defined to be 20 timesteps, which is more or less equal to the defined 10 seconds. This is the least amount of time that the agent can, or cannot, pace the runner.

$$A_t : \{0, 20, 25, 30, 40\}$$

In the set A_t the values for 25, 30 and 40 means respectively the amount of timesteps that the agent can select to pace the runner. These values were arbitrarily chosen to have a set of values relatively distributed between the 10 and 20 seconds of keeping the auditory pacing activated.

For the rewards R_t, we need to provide rules for the agent to learn how to balance the amount of pacing provided. The rules for the rewards are the following

$$R_t \begin{cases} s_t > 1.5\% & \textbf{-1} \\ \text{If } A_t == 0 & \begin{cases} s_t < 0.2\% & \textbf{+1} \\ \text{else,} & \textbf{0} \end{cases} \\ \text{If } A_t > 0 & \begin{cases} |s_{t+1} - s_t| < 0.2\% & \textbf{-1} \\ \text{else,} & \textbf{+2} \end{cases} \end{cases}$$

When the new difference with respect to the target pace ($s_t \in S_t$) is higher than a 1.5% means that, no matter the action taken, the runner is too far from the target and thus the reward is a -1. When the agent decides not to pace ($A_t == 0$), if the s_t is less than a 0.2% from P means that the runner closely sticking to the target pace and thus a reward of $+1$ is given. If the runner is between 1.5% and 0.2% of P, means that the runner is still close to the target pace without diverging too much and thus the reward is 0. On the contrary, we have other rewards when the agent decides to pace ($A_t > 0$). If the pace is activated and after the pacing block the different between the new state s_{t+1} and the previous state s_t is less than 0.2% means that the pacing did not work or was not necessary and thus a reward of -1 is given. Otherwise, means that the pacing is affecting the behavior of the runner and thus a reward of $+2$ is given.

The default learning episode is set to consist of 16 action blocks (roughly over 400 timesteps) which are is the average amount of action blocks that we can obtain from the experimental data. The first action is set to be 0 in order to get a proper estimation of the initial average pacing and no reward is given. From this point onwards the states, actions and rewards work as described.

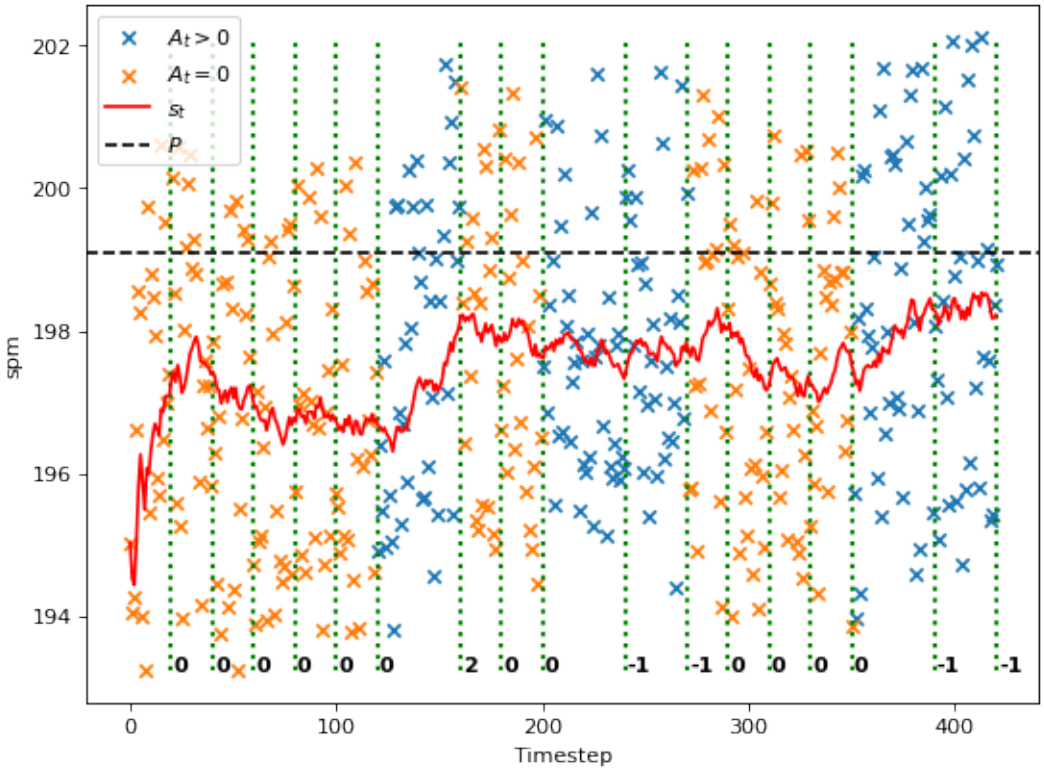

Figure 3.9: Example of an episode with a final reward of *-1*

In figure 3.9 an example of an episode generated using the defined environment can be observed. The green vertical dotted lines represent the end of an action block, after each of these action blocks the reward r_t is shown at the bottom. The orange $\times$'s represent the not paced timesteps ($A_t == 0$)

while the blue ones are the ones in which the pacing is active ($A_t > 0$). The dashed horizontal line represents the target pace P and the solid red line is the state s_t. The final accumulated reward for this episode $\sum r_t$ is -1.

The defined MDP and its respective reward function were chosen after a number of iterations alongside with an implementation of a RL algorithm. Each time a new definition was done, it was tested trying to make the RL model to learn from it. As the goal is to minimize the amount of times that the auditory pacing is activated, it is important to adjust the balance between the positive and negative rewards. If this is not done, it could result in the agents learning a high probability of how not to pace (e.g. 95% of the time auditory pacing and 5% of the time on no matter the current state).

3.3.2. Heuristic algorithm

This is a simplistic solution for the pacing problem in which we are only taking into account the measured step frequency. Nevertheless, in the future, the same pacing model can include in the state different inputs that can be measured from the runner as the heart rate, temperature or oxygen intake. This presented pacing model can find a good solution by using a heuristic approach in which are defined some boundaries for the pace and when trespassing these, the auditory pacing is activated. The algorithm 1 depicts this described logic.

Algorithm 1: Heuristic algorithm

Input : s_t
Output : A_t

if $s_t > 2.7\%$ **then**
  $A_t = 40$
else if $s_t > 2.2\%$ **then**
  $A_t = 35$
else if $s_t > 1.5\%$ **then**
  $A_t = 30$
else if $s_t > 1.1\%$ **then**
  $A_t = 20$
else
  $A_t = 0$
end if
return A_t

In figure 3.10 can be seen an example of the heuristic algorithm interacting with the environment.

3.3.3. Environment reference values

The average total reward and standard error for 500 episodes using the heuristic approach resulted to be 3.63 ± 3.66. This value is used as reference for the following learning algorithms to consider the environment as solved.

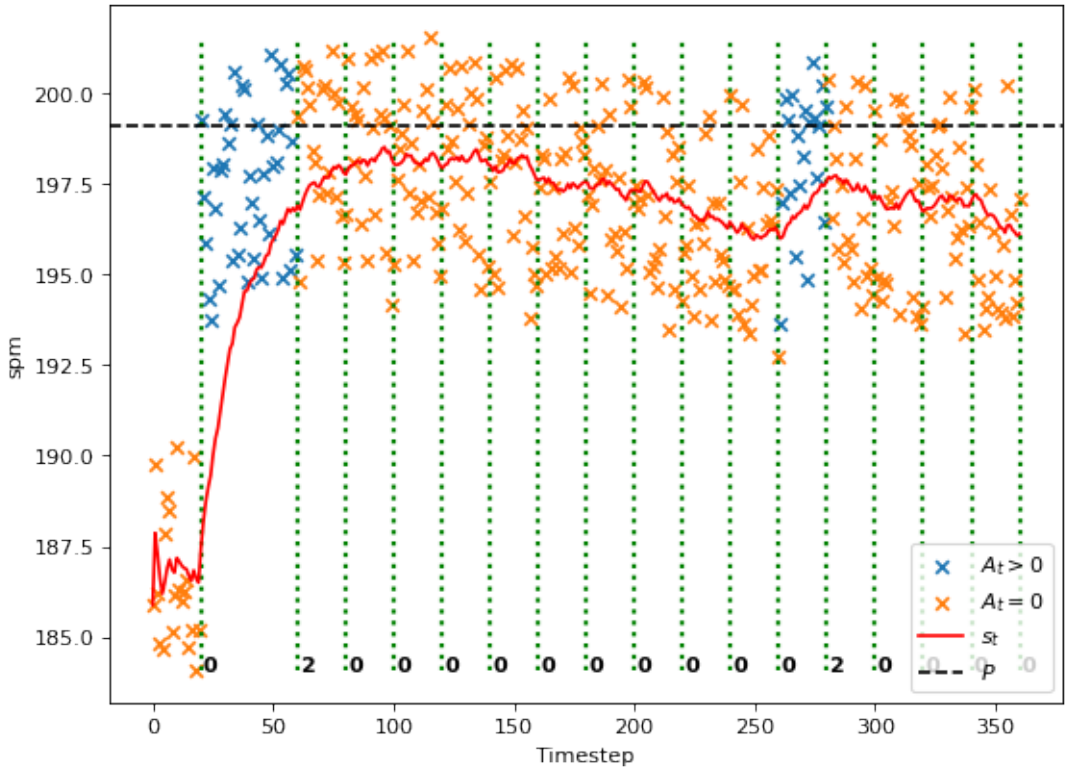

Figure 3.10: Example of an episode using the heuristic algorithm having a final reward of *+4*

As additional reference values for the environment, we have the random behavior with actions A_t chosen from an uniform distribution with an average reward of -5.704 ± 3.66. Then we have the *non-pacing policy* that is when the agent lets the runner to run without activating the auditory pacing ($A_t = 0$) with an average reward of -12.184 ± 6.125. And the *always-pacing policy* where the agent keeps the auditory pacing on ($A_t > 0$) for the whole episode with an average reward of -8.376 ± 3.25.

4

Experiments

Having in mind the research questions in chapter 1.1 and using the tools described in 3, a set of experiments were designed and are described in this chapter.

Initially, to answer the research question 1.1.1(I), some of the algorithms described in chapter 2 were implemented. These were tested initially in a default environment, the cart-pole [34], before being used in the pacing environment, described in the section 3.3. In order to verify if the pacing is efficient or not, the results are compared with the heuristic algorithm described in 3.3.2.

Afterwards, to answer the research question 1.1.1(II), an experiment using different batch sizes with randomly sampled data (sampling policy) generated with the simulator was run.

Thereafter, it is explained how from the provided experimental data we create MDP transitions to be used as initial batch for the training of the RL algorithms. Then, to answer the research question 1.1.1(III), another experiment was run using as input sampling policy these generated transitions from the experimental data.

Finally an experiment to compare the results from the Off-PAC and OPPOSD algorithms using both data from the simulator and from the experiments as input batch was run.

4.1. Learning algorithms

The definition of the pacing environment in 3.3 depended on its ”learnability”. For this reason, alongside with the definition of the MDP it was necessary to keep testing it. For this purpose, implementations

of the learning algorithms RL [27] and Actor-Critic [28] defined in the section 2.2.7 and the algorithms Off-PAC [11] and PPO [32] described in section 2.2.8 were implemented.

These algorithms were implemented in an incremental way towards the goal of implementing of the OPPOSD algorithm [4]. As most of the implementations are designed to work in an online approach, their preliminary performances and behaviors are compared using online updates. All of the algorithms are tried both in the cart-pole environment [34] and the pacing environment.

Afterwards, an adjustment to the policy updates is done to evaluate the performance of the batch off-policy approach for the Off-PAC and the implemented OPPOSD algorithms.

4.1.1. Online learners

The framework defined in Appendix A was implemented, in order to have a better control over specific parts of the learning process, like the definition of the experiments or the learners, without having to change anything else from the code.

The way in which the algorithms interact with the environment in an online way is depicted in the figure 4.1. In the diagram the flowchart of the actions taken can be seen. In the first step a set of F transitions are gotten by directly interacting with the environment using the policy π_0. These actions are then stored in a replay buffer. Unless the algorithm being used follows an off-policy approach, then these F steps are used to directly update the policy π_0 with a policy gradient update. In case the algorithm being used follows an off-policy approach (i.e. The Off-PAC or the PPO) then, after filling the replay buffer a set of minibatches are sampled from it. These minibatches are later used to keep updating the initial policy π_0 with off-policy gradient updates, and thus result in a new policy π for each step. While using off-policy algorithms in an online approach, we are trying to have a better sampling efficiency and thus learning the most possible from the available data.

The implemented algorithms for testing were the initial reinforce, the actor-critic, the Off-PAC and the PPO. These were tested using the cart-pole environment [34]. In this environment the agents need to balance a mass attached to a pole in an upright position, by applying one of two sideways movements to a cart on a frictionless track. The horizon has a maximum of 200 timesteps and the rewards are defined as $+1$ for each timestep that the pole is standing.

In figure 4.2 the averaged results for 5 runs from the implemented algorithms in the cart-pole environment can be seen. The raw results are also averaged in windows of 10 timesteps with the purpose of smoothing the graph. Additionally, for each line the standard error can be seen, and this can be considered as a measurement of the stability. The interactions with the environment were limited to 1000, each one of them consisting of a sample of 1000 environmental transitions ($F = 1000$). All of these algorithms were implemented with a model of a neural network with two 128-unit hidden layers

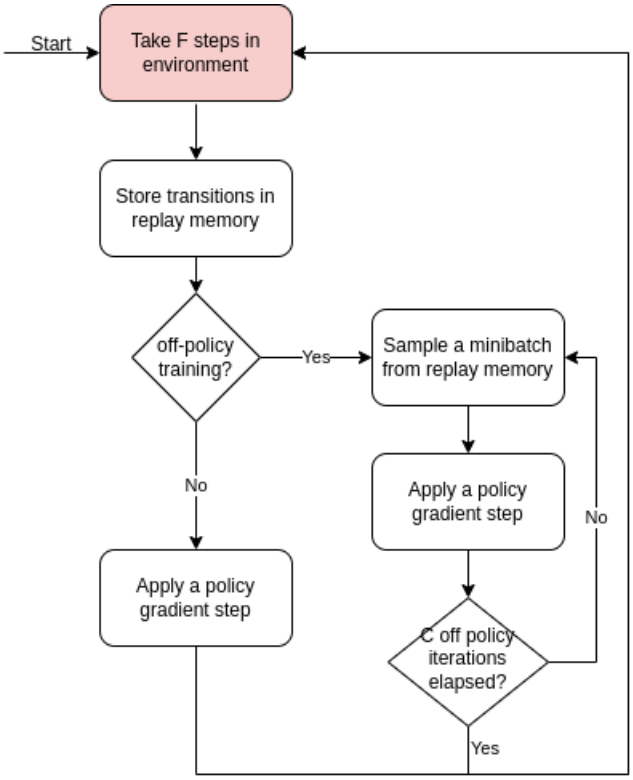

Figure 4.1: Flowchart of the way the RL algorithms interact with the environment in an online approach

each followed by a ReLU layers. The Adam optimizer with a learning rate of $5e-4$ was also used. The neural networks for the actor-critic, Off-PAC and PPO implementations have an additional output value to be used as the critic output (instead of using an independent Neural Network for the critic). Both the Off-PAC and the PPO had additional 128 off-policy iterations ($C = 128$). The PPO clipping threshold value was set to 0.2. The exploration ϵ started in 1 and ended in 0.1 with an anneal time of 2. In the figure 4.2 can be seen that the off-policy approaches are the ones that require the least amount of data in order to learn a policy that performs better that the random policy. The base reinforce algorithm presents a good performance but can become unstable with time. The actor-critic takes longer to learn but its output results tend to be more stable. The Off-PAC learns fast, but is the algorithm that presents the most instability. This instability can be caused by the disregard of the IS term in the equation 2.20 [29]. And finally, the PPO appears to be the fastest and most stable learner in this environment.

After having these algorithms implemented and working in the cart-pole environment, they were put in the pacing environment in order to test its ”learnability”. In figure 4.3 can be seen the performances of the algorithms in the pacing environment. The hyper-parameters are the same as before except the anneal time that now is $5e3$ in order to promote the exploration. The pacing environment is more sensitive to exploration than the cart-pole given the distribution of the states and rewards. In this environment, for a batch of $1e5$ transitions sampled with a random policy, the probability of getting a

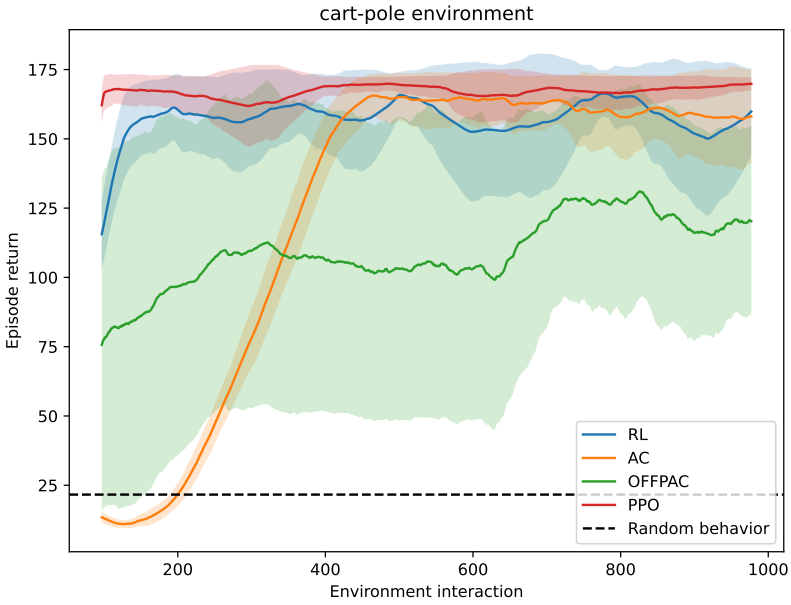

Figure 4.2: Behavior of base reinforce, actor-critic, Off-PAC and PPO algorithms in the cart-pole environment

positive reward ($r_t > 0$) is of 17.24%. Of this 17.24% a 42.28% of the positive rewards correspond to the action of not pacing ($A_t = 0$) while the remaining 57.72% are pacing actions ($A_t > 0$). Additionally the 72.09% of the transitions in the batch are negative rewards of whom only the 2% correspond to the action $A_t = 0$. the remaining 10.74% of the batch transitions have a $r_t = 0$ which is only possible to get with the action $A_t = 0$. For these reasons, if the exploration is not done properly (anneal time too small), the algorithm could end up generalizing and converging to the non-pacing policy.

The table 4.1 shows a summary of the hyper-parameters used for both environments. These values were selected to resemble the most possible to the hyper-parameters

It can be seen in figure 4.3 that the algorithms are able to solve the environment within the 1000 interactions with the environment (In total $1e6$ transitions). The Off-PAC and the PPO, being the only two off-policy approaches, show a better sampling efficiency than the other two, in particular the PPO.

Hyper-parameters	Cart-pole	Pacing
NN layers	2×128	2×128
Learning rate (actor-critic)	$5e-4$	$5e-4$
Batch size (F)	1000	1000
Off-policy iterations (C)	64	64
ϵ_{max}	1	1
ϵ_{min}	0.1	0.1
Anneal time	2	$5e3$

Table 4.1: Online hyper-parameters used for the cart-pole and Pacing environments

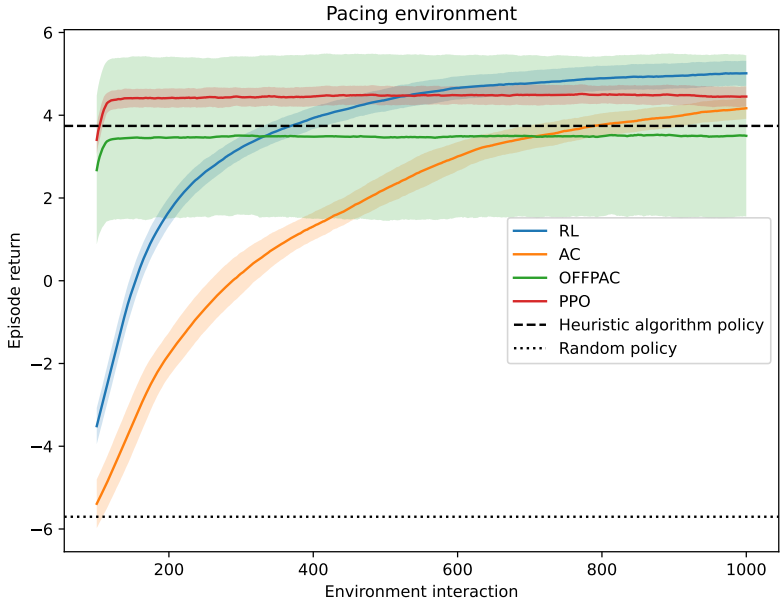

Figure 4.3: Behavior of base reinforce, actor-critic, Off-PAC and PPO algorithms in the pacing environment

4.1.2. Batch Off-policy learners

The idea now is to limit just to one the sampling from the environment in figure 4.1. Now the flowchart will look like the one in the figure 4.4. In here we sample an initial group of transitions with an initial sampling policy μ. This initial group of transitions is going to be then the transition batch from which we are going to sample mini-batches to train the algorithms in an off-policy way.

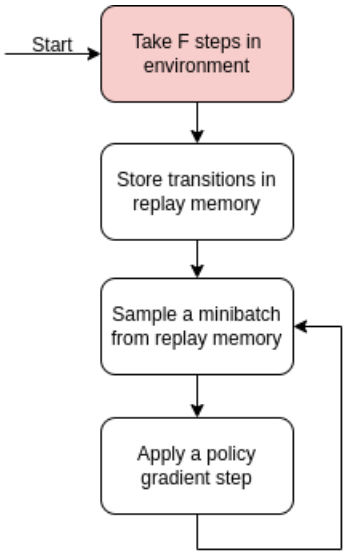

Figure 4.4: Flowchart of the way the RL algorithms interact with the environment in an batch off-policy approach

As suggested in [4] the sampling policy used to gather the data is assumed to be stochastic, because

if it is deterministic, the algorithm will not be able to estimate the performance of any other policy. When taking the batch off-policy approach, the IS term, which plays an important role in the PPO algorithm, tends to be constantly clipped due to the constantly growing ratio between the sampling policy μ and the target policy being learned π. For this reason, the only algorithms taken into account for this approach were the Off-PAC and the OPPOSD. The implementation of these algorithms was done by trying to resemble best as possible the described in [29] and [4]. For this reason three different learning models were used, one for the actor, one for the critic, and a final one for the estimation of the $w(s)$ function that tries to approximate the IS value. All of them consisted of neural networks with two 128-unit hidden layers each followed by a ReLU layers except for the $w(s)$ which had the last activation function $\log(1 + \exp(x))$ to guarantee that $w(s) > 0$ for any input. To estimate the performance of the policy π at a given timestep t, this was evaluated in the simulator for 10 episodes after each 10 policy gradient steps. This way the average return per episode is estimated.

As in [4] a batch of approximately $1e5$ transitions was sampled from the cart-pole environment using a random sampling policy μ with an average reward of 22. These transitions were stored in a transition batch from which mini-batches of $5e3$ were sampled for each of the off-policy gradient steps. The critic model was updated 10 times for both Off-PAC and OPPOSD before each actor policy's update. For the OPPOSD algorithm, the $w(s)$ function was updated 50 times before each actor policy's update. The Adam optimizer with a learning rate of $1e^{-3}$, as suggested in [4]. As this is a batch off-policy approach, there is no need of adding any additional exploration probabilities to the output of the model. The table 4.2 presents summary of the hyper-parameters used for the batch off-policy scenario.

Hyper-parameters	cart-pole
Batch size (F)	$1e5$
Minibatch size	$5e3$
NN layers	2×128
Learning rate (actor-critic)	$5e-4$
Learning rate (w)	$1e^{-3}$
Critic iterations per policy update	10
w iterations per policy update	50

Table 4.2: Batch off-policy hyper-parameters used for the cart-pole environment

In the figure 4.5 can be seen the average behavior for 5 runs for $5e3$ policy gradient steps for the Off-PAC and the OPPOSD algorithms. It can be seen here that both of the implementations are able to learn from the randomly sampled batch and get results averaging the 200 after approximately 2500 gradient steps.

Batch off-policy algorithms in the pacing environment

In order to test the behavior of the Off-PAC and the OPPOSD algorithms, we tried them on the pacing environment having again a batch of approximately $1e5$ transitions sampled from the environment using

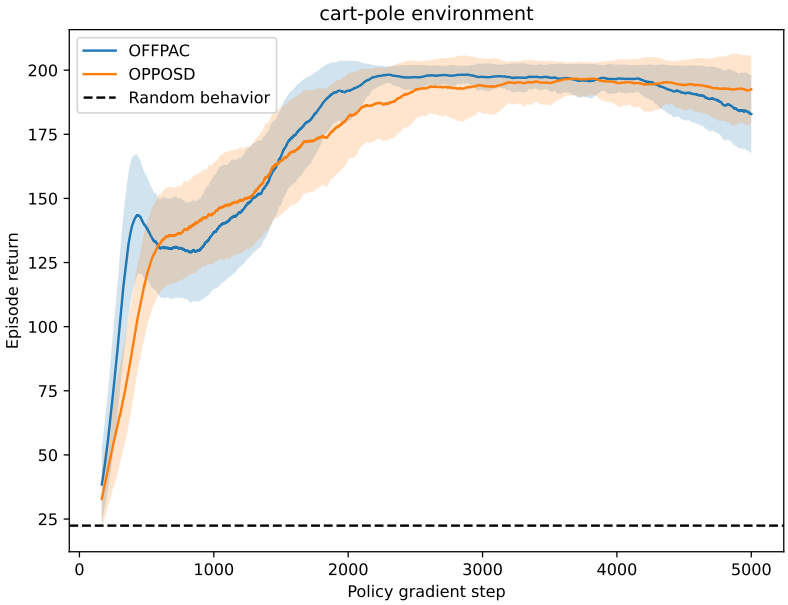

Figure 4.5: Behavior of the batch off-policy algorithms in the cart-pole environment

a random sampling policy μ. And again, from these transitions mini-batches of $5e3$ were sampled for each of the off-policy gradient steps. All the other hyper-parameters of the algorithm remained unchanged as in table 4.2.

In figure 4.6 can be seen the average behavior for 5 runs for $5e3$ policy gradient steps for the Off-PAC and the OPPOSD algorithms with their respective error. For both the OPPOS and the Off-PAC were used the same 5 randomly generated datasets. It can be seen in the figure 4.6 that OPPOSD seems to be slightly more stable (less variance) in improving the target policy compared to the Off-PAC.

4.2. Available data impact in the learning

In order to determine the amount of data necessary for the batch off-policy algorithms to learn it was decided to test the Off-PAC and the OPPOSD using different batch sizes. The experiment was run using three different learning models, one for the actor, one for the critic and a final one for the estimation of the $w(s)$ function as described in the section 4.1.2. Additionally, the size of the mini-batches was $1e3$, except when the batch size was of $2e3$, then the mini-batch size was changed to 500. The list of hyper-parameters used is shown in the table 4.3.

In order to estimate the performance of the target policy π, it was tested in the simulator for 10 episodes after each 10 policy gradient steps. This way the average return per episode is estimated. All the experiments were run in the HPC cluster from the TU Delft university.

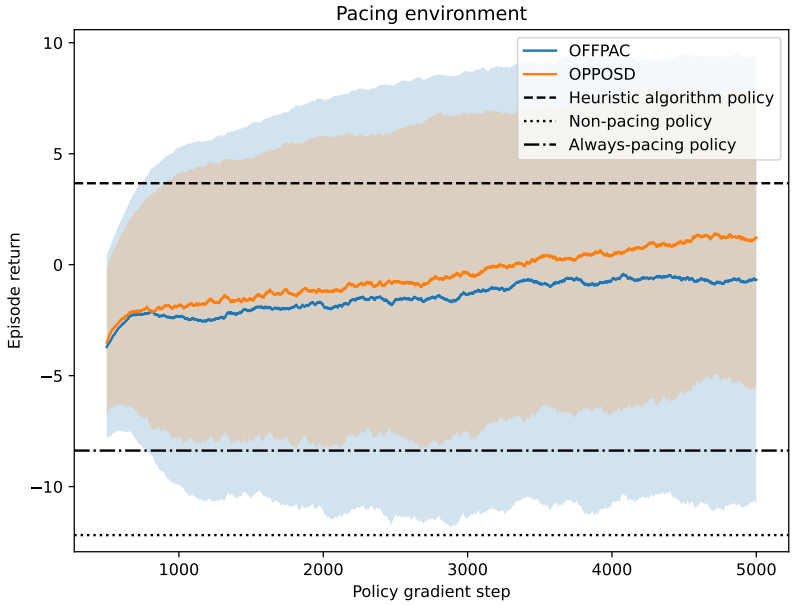

Figure 4.6: Behavior of the batch off-policy algorithms in the pacing environment

Hyper-parameters	Pacing
Minibatch size ($F > 2e3$)	$1e3$
Minibatch size ($F = 2e3$)	500
NN layers	2×128
Learning rate (actor-critic)	$5e-4$
Learning rate (w)	$1e-3$
Critic iterations per policy update	10
w iterations per policy update	50

Table 4.3: Batch off-policy hyper-parameters used for the pacing environment experiment varying the batch size F

The figure 4.7 shows the average result of 5 runs using the Off-PAC implementation with 4 different batches of sizes $2e3$, $5e3$, $6e3$ and $24e3$ (intermediate values had similar behaviors). All of these batches were generated using a random sampling policy μ in the simulator and for each single run was generated a new batch was generated. In general can be seen that for the Off-PAC algorithm the results have a high variance. This supposes that the initial batch used has impact in how the algorithm performs. Furthermore, the batch size appears to have a slight influence in the mean but not in the variance. There are some specific randomly generated batches that can lead the algorithm to find a relatively good solution while some others make the algorithm to converge to the always-pacing or non-pacing policies.

The figure 4.8 shows the average result of 5 runs using the OPPOSD implementation with 5 different batches of the same sizes as before: $2e3$, $5e3$, $6e3$ and $24e3$ (intermediate values had similar behaviors). All of the batches (20 in total) were re-utilized from the Off-PAC experiment. Even though for the

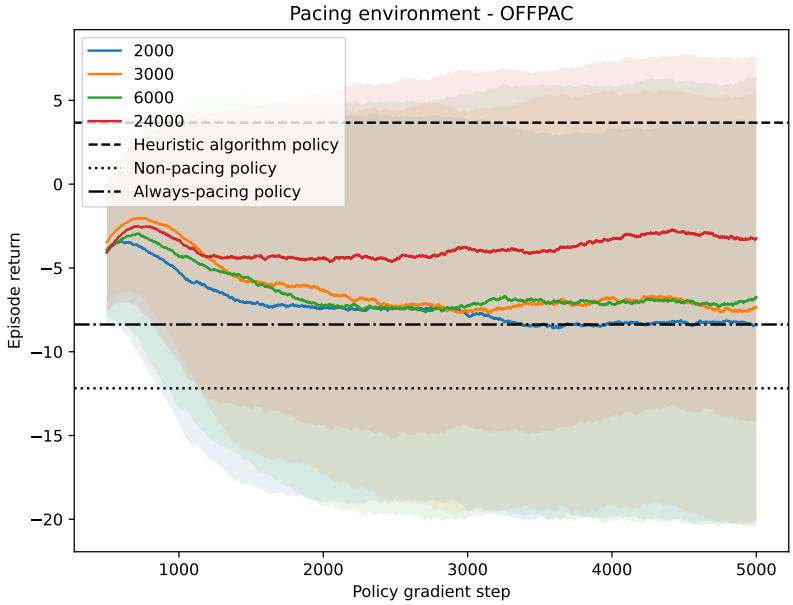

Figure 4.7: Impact of the initial batch size on the batch Off-PAC in the pacing environment using a stochastic policy with the simulator

experiments with $3e3$ and $24e3$ the OPPOSD algorithm appears to have similar results to the Off-PAC, for the batches of $2e3$ and $6e3$ seems to have better results. It can also be noticed that for some of the batches of $2e3$ and $6e3$ the OPPOSD can score even higher than the best Off-PAC score that learns above the heuristic algorithm's average reward.

4.3. Behavior Policies

The training batches used for the batch off-policy approaches could be generated with different known sampling policies π_0. Nevertheless, as suggested in [4] these sampling policies are assumed to be stochastic, otherwise, the algorithm will not be able to estimate the performance of any other policy. A preliminary experiment showed that using a trained policy as the sampling policy caused the learning algorithms to converge to the non-pacing policy. For this reason the two sampling policies that were to be tried are the random transitions generated from the simulator and the transitions generated from the provided experimental data described in the section 3.1.

4.3.1. Experiments training batch

In order to use the data from the experiments described in section 3.1 as a batch of transitions, it is necessary to translate them into MDP-like transitions. For this purpose, the input files are divided into action blocks according to the MDP of the pacing environment.

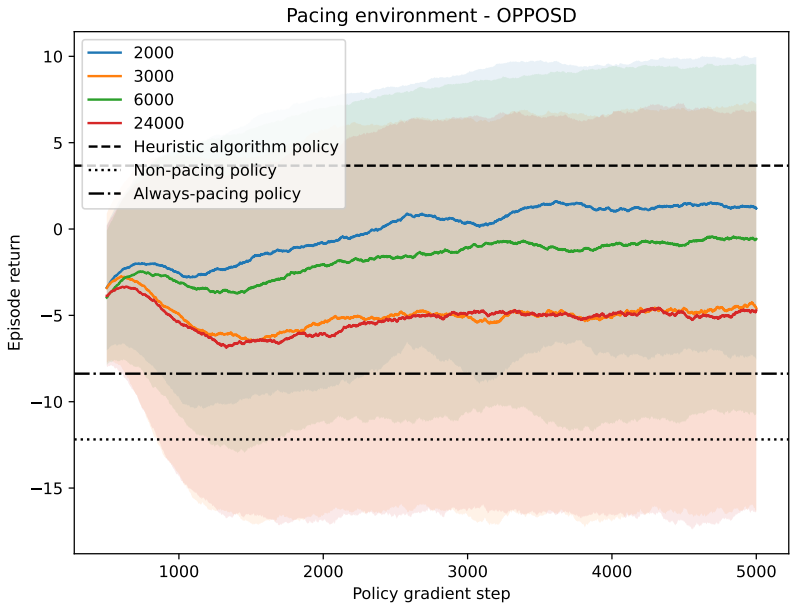

Figure 4.8: Impact of the initial batch size on the batch OPPOSD in the pacing environment using a stochastic policy with the simulator

In figure 4.9 some examples of the translation process can be seen. If the action block starts with the pacing activated (blue $\times$'s) then the transition is going to have an A_t randomly chosen among the $A_t > 0$ actions and the block will last accordingly with the time of the action. Otherwise, if the pace is not active (orange $\times$'s) then $A_t = 0$ and the action block will last an amount of 20 timesteps. The rewards obtained are calculated following the R_t logic described in 3.3.1. This calculation is performed taking into account the initial pace at the beginning of the action block and the final pace at the end of the same action block.

After translating the 66 available experiments, we get approximately the amount of 1012 transitions to be tested in the environment. We can regenerate the translation of the experiments in order to get more transition for the training process.

4.4. Batches from experimental data as initial batch

In order to now test the impact of replacing the data from the simulator with a different policy, the experimental data was translated as described in 4.3. From the experimental data we get approximately $1e3$ transitions per conversion, as explained in the section 4.3.1. These conversions had some random components regarding the pacing actions taken (20, 25, 30 or 40). This small randomness can generate slightly different transitions per run. For this reason, the impact of re-generating new transitions out of the same set of experiments was also part of this test. In this case, $2e3$ is the approximate size for the

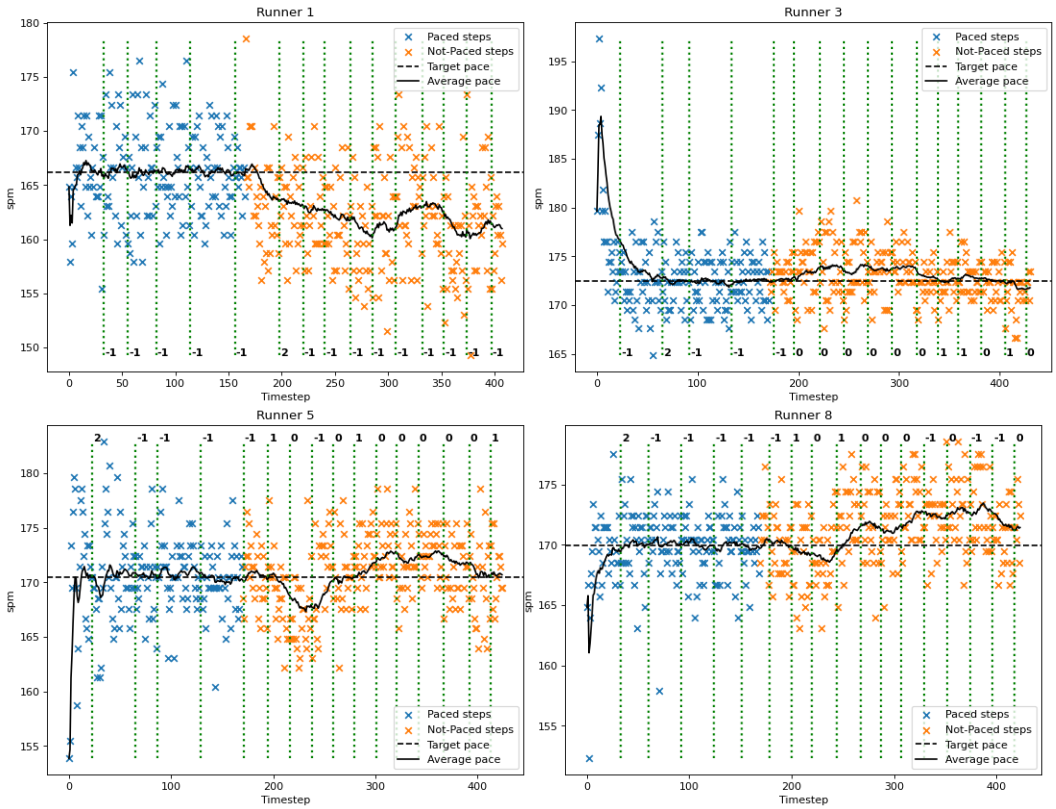

Figure 4.9: Example of experiments from runners translated into MDP transitions

transitions generated from the experimental data twice and in a similar way for the values $3e3$, $6e3$ and $24e3$.

Figures 4.10 and 4.11 show the result of the experiments for the algorithms Off-PAC and OPPOSD respectively. For the Off-PAC experiment seems that none of the used batches have a big impact in the result. The Off-PAC could be considered to not be able to improve the target policy. On the other hand, the OPPOSD algorithm shows a better use of the experimental data. Here again none of the batch sizes seem to have a big influence in the general behavior and even for the batch size of $2e3$ seems that the target policy improves.

4.5. General comparison

Given the previous results a new test using the least possible amount of data from the experiments (Batches of approximately $2e3$ transitions) and from the simulator was done. For the experimental transitions the initial batch was generated similarly to the previous test. For the simulated data the randomly generated dataset that worked best for the OPPOSD algorithm, in the experiment shown in figure 4.8, was used to achieve the best possible performance. For each combination (OPPOSD, Off-PAC and experimental, simulated data) the test was run 5 times. The results are shown in the figure

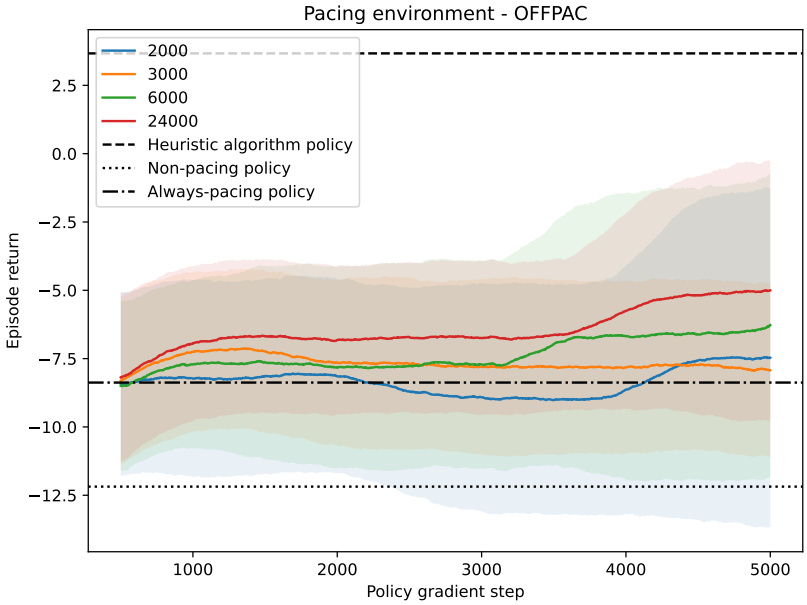

Figure 4.10: Impact of the initial batch size on the batch Off-PAC in the pacing environment using the experimental data converted into transitions

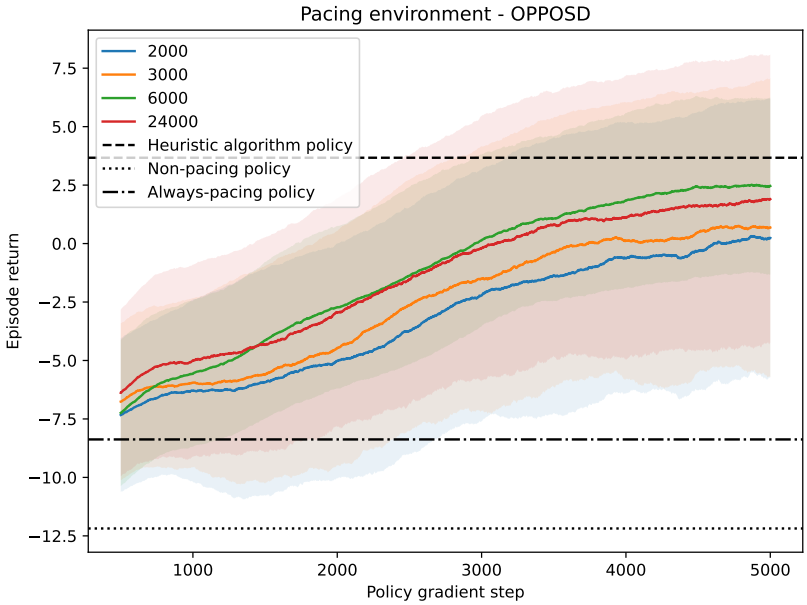

Figure 4.11: Impact of the initial batch size on the batch OPPOSD in the pacing environment using the experimental data converted into transitions

4.12 and are run up to $1e4$ policy gradient steps.

In the figure 4.12 can be appreciated with red tones the experiments done with the Off-PAC algorithm and with blue tones the experiments done with the OPPOSD. In red is the Off-PAC using the batch

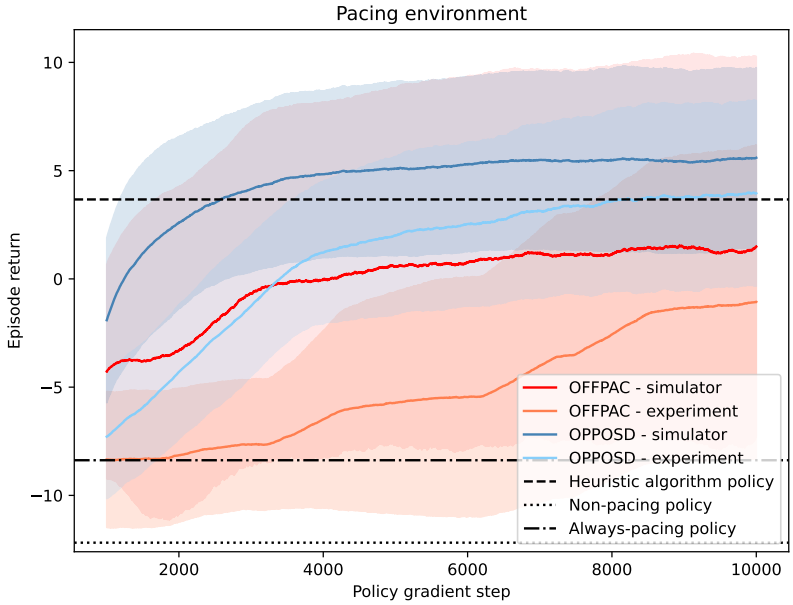

Figure 4.12: Compared behavior of the off-policy algorithms (Off-PAC and OPPOSD) using as input data a randomly generated batch from the simulator and the translated experimental data. Both batches had a size of $2e3$ transitions

generated with the simulator. The Off-PAC seems in general to improve the target pacing policy but it has a lot of instability. In dark blue is the OPPOSD using also the batch from the simulator. This approach seems to be the best overall to find a relatively good pacing policy. Then, in light blue is the OPPOSD algorithm using the batch generated from the experimental data. This one also improves the target policy even though does not match the final scores form the OPPOSD using the simulated data. Finally, the red coral line represents the Off-PAC using the experimental data as input batch. This one specifically seems to be the one that performs the poorest of them all.

5

Discussion

5.1. General comments

As seen in the experiments from chapter 4 both the Off-PAC and OPPOSD have a similar behaviors when using as initial batch a set of transitions generated directly from the simulator using a stochastic sampling policy μ. This behavior can be observed as for the cart-pole environment (figure 4.5) as for the pacing environment (figure 4.6). Despite the OPPOSD algorithm showing generally results with slightly less variance than the Off-PAC, these were not the expected differences as seen in [4], specially for the cart-pole environment. On the other side, when the probabilities of the sampling policy are not completely clear, as the ones obtained from the experimental data, we can observe that the additional IS term estimator of the OPPOSD algorithm has a more noticeable impact. This can be noticed comparing the behaviors from the figure 4.10 and the figure 4.11. For the OPPOSD figure, it is clear the upwards trend of the learned target policies, while for the Off-PAC that is not clear.

For the OPPOSD (figure 4.8), the upwards trend of the learned target policies is clear, while this is not the case for the Off-PAC (figure 4.7).

In the figures 4.7 and 4.8 it can also be noticed how the performance of the algorithms seems to be affected by the input batch. The high variance tell us that for some generated batches the algorithm finds a pacing policy as good or better than the heuristic algorithm but for some others the algorithm converged to the non-pacing or the always pacing policies.

In general, can be said that the size of the input batch does not have a big impact in the resulting policy but what matters is the distribution of this data and the way in which the algorithms samples from it. While it is not possible to change the way in which the algorithm samples the data (due to the mathematical suppositions described in in [4]) a further statistical analysis could be made to verify which are the specific characteristics of a dataset for the OPPOSD algorithm to learn a good pacing policy.

5.2. Answer to research questions

In chapter 4 we designed the experiments to answer the research questions presented in chapter 1. Based on the results obtained we provide the following possible answers:

1.1.1(I) *Can a trained Deep Reinforcement Learning algorithm learn a policy to provide auditory pacing efficiently?*

The pacing environment was designed to quantify the performance of certain policies in the pacing regulation problem. Besides, the heuristic algorithm was chosen as a good alternative to provide an efficient pacing strategy. We consider that if the final policy reached by a reinforcement algorithm trained in the pacing environment outperforms the heuristic algorithm, then we could say that the policy provides a relatively efficient auditory pacing. Most of the online-learning algorithms implemented (RL, AC, Off-PAC and PPO) were able to find a relatively efficient pacing policy, as seen in the figure 4.3. However, For the batch off-policy algorithms (Off-PAC and OPPOSD) presented in figure 4.6 it was in general harder to find an efficient pacing policy. Using these batch off-policy algorithms we get higher standard errors, so some of the experiments improved the target policy while others did not. This results made us realize that the success of these learning algorithms was closely related to the initial sampled batch. Furthermore, from the figure 4.11 we can see that using experimental data the OPPOSD algorithm is steadily improving the pacing policy towards a relatively efficient pacing policy. In the end, in figure 4.12 it was shown that when the correct input batch is used, it is possible to find relatively efficient pacing policies using the proposed batch off-policy algorithms.

1.1.1(II) *What is the least amount of training data necessary for the implemented Deep Reinforcement Learning algorithms to learn a relatively good pacing policy?*

To answer this question, we tested the algorithms with different batch sizes (section 4.2 and 4.4). In section 4.2 random data from the simulator was used to generate many initial batches of different sizes. It can be seen in figure 4.7 that for the Off-PAC algorithm the error is very high and that the batch size does not really have a big impact in the performance. Even with an initial batch size of $2e3$ for the Off-PAC algorithm there were batches that leaded the algorithm into a relatively efficient pacing

policy. For the OPPOSD algorithm, in figure 4.8, we see a similar behavior as for the Off-PAC, the main difference is that the average score for the resulting policies is a bit higher. This behavior might suppose that the OPPOSD algorithm could lead in general to better pacing policies especially for the $2e3$ batch size. In the section 4.4 the initial batches were generated from the data of the experiments with real runners. From this data we could generate around a $1e3$ transitions adding a small random component that let us re-generate the transitions having slightly different values. This way we created again batches sets with sizes of $2e3$, $3e3$, $6e3$ and $24e3$. With these initial batches we tried the Off-PAC algorithm but the results, seen in figure 4.10, showed that the policies in general converged around the always-pacing policy no matter the batch size. On the other side, the OPPOSD algorithm showed a better performance using the experimental data. In figure 4.11 we see that no matter the batch size, the OPPOSD algorithm keeps constantly improving towards a relatively efficient policy.

From these experiments the minimum amount of data from which we were able to lead the RL algorithms into a relatively efficient policy turned out to be initial batches of approximately $2e3$ transitions. In figure 4.12 all experiments are done using initial batches of $2e3$ and it can be noticed that the OPPOSD algorithm is able to find relatively efficient policies for each of the batch types. The batch size of $2e3$ transitions could be equivalent to 15.556 hours of experimental data with real runners assuming that the average transition time is $28s$.

1.1.1(III) *Is it possible to use experimental data from real runners as training data for the implemented algorithms to learn a relatively good pacing policy?*

Experiments from section 4.4 and 4.5 helped us also to answer this question. In figure 4.11 we see that the OPPOSD algorithm using experimental data is able to improve the target policy in terms of efficiency. Additionally, in figure 4.12 we can see that the OPPOSD algorithm with an initial batch of $2e3$ transitions from the experimental data is able to improve the pacing target policy until matching on average the heuristic algorithm performance after $1e4$ policy gradient steps. These results are promising and tell us that using the OPPOSD algorithm is possible to find a policy that is relatively good in terms of efficiency when comparing with the heuristic algorithm's performance.

1.1.1(IV) *How could the reached solution be deployed for real runners?*

The resulting actor model is a trained neural network that has as input space the current state and as output the probabilities of taking an action. For the performed experiments the used model had 2 fully connected layers of 128 units. This model could be deployed as part of an integrated application (mobile phone or embedded in the headphones) that measures the current running pace of the user (using accelerometer sensors) and based on the model's output, the auditory pacing could be activated or not. Through the deployment of this application the real efficiency of the resulting policy could be tested with real runners.

Additionally, a more interesting implementation would be to deploy the actor model alongside with any of the online learning algorithms (RL, AC, Off-PAC, PPO, etc.). This way the algorithm could keep learning using the data gathered from each run with the real runners. This online reinforcement learning implementation could eventually find target policies that are personalized for every runner.

5.3. Limitations

One of the limitations we had in the current work could be the length of the experimental data. The average experiment with the real runners lasted approximately 2.5 minutes, which limited the size of the translated episodes to only 16 transitions on average. Longer experiments could have also given us a better understanding of the response behavior for some runners to the auditory pacing. Another incertitude that we had along the project was the minimum required amount of time required for the runner to synchronize with the auditory pacing and it was assumed to be around 10 second. Perhaps a new set of experiments with real runners could help us to have a better idea about these constrains (i.e. time required to synchronize, response to intermittent pacing, maximum amount of time that a runner can keep the target pace, etc.).

Additionally, experiments measuring more variables (i.e. heart rate, amount of oxygen intake, temperature, etc.) could help us to extend our model to be more precise. However, this could also imply a much more complicated simulator.

6

Conclusion

In this document, we were able to implement the OPPOSD algorithm [4] and train it using batches of experimental data to find an efficient policy capable of providing intermittent pacing to runners. To learn a relatively good target policy using Reinforcement Learning algorithms, a high sampling efficiency must be guaranteed. The batch off-policy algorithms like Off-PAC [11] and PPO [32] try to provide a good sampling efficiency. Nevertheless, the infinite horizon while learning presents a problem due to the continuously increasing difference between the sampling policy and the target policy being learned. Algorithms that tackle this infinite horizon problem for batch learning like the OPPOSD are the most recommended in this regard. Also, for these algorithms to be trained and to perform, is necessary to define a proper MDP that represents the targets of the environment e.g., efficient intermittent pacing. The MDP together with a simulator are important tools for estimating the performance of the policies being learned at any point in time. The results showed that for the algorithm to learn a relatively good policy, it is needed at least an amount of approximately $2e3$ transitions from the MDP. These transitions are directly obtained and interpreted from experiments done with real runners on a treadmill. The obtained results are compared against a heuristic algorithm that allows us to define an efficiency measure for the provided pacing. In this example, rather simplistic, we only used one input (estimated steps per minute) to calculate the current state of the environment, but in the future, the input state dimension can be widened. Inputs like the runner's heart rate, intake of oxygen, or stress level could be used as inputs for the model. Nevertheless, the simulator should also take into account these new variables.

The resulting trained policies are neural networks that can be implemented as complex as needed. The complexity may have an impact on the learning times but may perform better when learning in a more elaborated pacing environment. Additionally, the resulting trained weights of these neural networks could subsequently be deployed in real solutions like mobile applications or embedded in the headsets to provide the auditory pacing.

References

[1] C. (Lieke) E. Peper, Jolanda K. Oorthuizen, and Melvyn Roerdink. “Attentional demands of cued walking in healthy young and elderly adults”. en. In: *Gait & Posture* 36.3 (July 2012), pp. 378–382. ISSN: 0966-6362. DOI: 10.1016/j.gaitpost.2012.03.032. URL: https://www.sciencedirect.com/science/article/pii/S0966636212001270 (visited on 05/06/2022).

[2] Anouk Nijs, Melvyn Roerdink, and Peter J. Beek. “Cadence Modulation in Walking and Running: Pacing Steps or Strides?” en. In: *Brain Sciences* 10.5 (May 2020). Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, p. 273. ISSN: 2076-3425. DOI: 10.3390/brainsci10050273. URL: https://www.mdpi.com/2076-3425/10/5/273 (visited on 04/22/2022).

[3] Anouk Nijs, Melvyn Roerdink, and Peter J. Beek. “Effects of acoustically paced cadence modulation on impact forces in running”. en. In: *Gait & Posture* 90 (Oct. 2021), pp. 234–238. ISSN: 0966-6362. DOI: 10.1016/j.gaitpost.2021.09.168. URL: https://www.sciencedirect.com/science/article/pii/S0966636221004781 (visited on 04/21/2022).

[4] Yao Liu et al. *Off-Policy Policy Gradient with State Distribution Correction*. Tech. rep. arXiv:1904.08473. arXiv:1904.08473 [cs, stat] type: article. arXiv, July 2019. URL: http://arxiv.org/abs/1904.08473 (visited on 05/18/2022).

[5] Bryan C. Heiderscheit et al. “Effects of Step Rate Manipulation on Joint Mechanics during Running”. In: *Medicine and science in sports and exercise* 43.2 (Feb. 2011), pp. 296–302. ISSN: 0195-9131. DOI: 10.1249/MSS.0b013e3181ebedf4. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3022995/ (visited on 04/21/2022).

[6] George M Dallam et al. “Effect of a global alteration of running technique on kinematics and economy”. In: *Journal of Sports Sciences* 23.7 (July 2005). Publisher: Routledge, pp. 757–764. ISSN: 0264-0414. DOI: 10.1080/02640410400022003. URL: https://doi.org/10.1080/02640410400022003 (visited on 05/07/2022).

[7] Timothy J. Quinn et al. “Step Frequency Training Improves Running Economy in Well-Trained Female Runners”. en-US. In: *The Journal of Strength & Conditioning Research* 35.9 (Sept. 2021), pp. 2511–2517. ISSN: 1064-8011. DOI: 10.1519/JSC.0000000000003206. URL: http://journals.lww.com/nsca-jscr/Fulltext/2021/09000/Step_Frequency_Training_Improves_

Running_Economy.23.aspx?context=LatestArticles&casa_token=hQbT0uPKRz8AAAAA:G-LLj5cyn8FvnlVYqcXj9xqWbq97tdGNriAty7a6J_RG47UJ-oWS8-OSvg8M5cOS41x51d84KB0MI_ElPcED_lXOoJw (visited on 05/06/2022).

[8] Jeska Buhmann et al. “Shifting the musical beat to influence running cadence”. eng. In: *Expressive interaction with music : ESCOM 2017 conference proceedings*. 2017, pp. 27–31. URL: http://hdl.handle.net/1854/LU-8530848 (visited on 04/21/2022).

[9] Denis Yarats et al. “Improving Sample Efficiency in Model-Free Reinforcement Learning from Images”. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.12 (May 2021). Number: 12, pp. 10674–10681. ISSN: 2374-3468. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17276 (visited on 07/26/2022).

[10] Hamid Reza Maei et al. “Toward Off-Policy Learning Control with Function Approximation”. en. In: Jan. 2010, pp. 719–726. URL: https://icml.cc/Conferences/2010/papers/627.pdf (visited on 05/06/2022).

[11] Thomas Degris, Martha White, and Richard S. Sutton. “Off-policy actor-critic”. In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML’12. Madison, WI, USA: Omnipress, 2012, pp. 179–186. ISBN: 978-1-4503-1285-1. (Visited on 07/26/2022).

[12] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. en. In: *Proceedings of the 35th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2018, pp. 1861–1870. URL: https://proceedings.mlr.press/v80/haarnoja18b.html (visited on 07/26/2022).

[13] RICHARD BELLMAN. “A Markovian Decision Process”. In: *Journal of Mathematics and Mechanics* 6.5 (1957). Publisher: Indiana University Mathematics Department, pp. 679–684. ISSN: 0095-9057. URL: http://www.jstor.org/stable/24900506 (visited on 05/09/2022).

[14] Alvin W. Drake. “Observation of a Markov process through a noisy channel”. eng. Accepted: 2005-08-17T17:39:43Z. Thesis. Massachusetts Institute of Technology, 1962. URL: https://dspace.mit.edu/handle/1721.1/11341 (visited on 06/10/2022).

[15] Martin Dobiasch, Savvas Stafylidis, and Arnold Baca. “Effects of different feedback variants on pacing adherence in a field based running test”. en. In: *International Journal of Performance Analysis in Sport* 21.6 (Nov. 2021), pp. 1015–1028. ISSN: 2474-8668, 1474-8185. DOI: 10.1080/24748668.2021.1968662. URL: https://www.tandfonline.com/doi/full/10.1080/24748668.2021.1968662 (visited on 05/10/2022).

[16] Chris R. Abbiss and Paul B. Laursen. “Describing and Understanding Pacing Strategies during Athletic Competition”. en. In: *Sports Medicine* 38.3 (Mar. 2008), pp. 239–252. ISSN: 1179-2035. DOI: 10.2165/00007256-200838030-00004. URL: https://doi.org/10.2165/00007256-200838030-00004 (visited on 05/10/2022).

[17] Jutta Fortmann et al. “PaceGuard: improving running cadence by real-time auditory feedback”. In: *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services companion*. MobileHCI ’12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 5–10. ISBN: 978-1-4503-1443-5. DOI: 10.1145/2371664.2371668. URL: https://doi.org/10.1145/2371664.2371668 (visited on 04/21/2022).

[18] *RockMyRun, Music that Moves You*. en. URL: https://www.rockmyrun.com/ (visited on 05/10/2022).

[19] Mark te Brake et al. “Using beat frequency in music to adjust running cadence in recreational runners: A randomized multiple baseline design”. In: *European Journal of Sport Science* 0.0 (Feb. 2022). Publisher: Routledge _eprint: https://doi.org/10.1080/17461391.2022.2042398, pp. 1–10. ISSN: 1746-1391. DOI: 10.1080/17461391.2022.2042398. URL: https://doi.org/10.1080/17461391.2022.2042398 (visited on 05/10/2022).

[20] D. Wezenberg et al. “Mind your step: Metabolic energy cost while walking an enforced gait pattern”. en. In: *Gait & Posture* 33.4 (Apr. 2011), pp. 544–549. ISSN: 0966-6362. DOI: 10.1016/j.gaitpost.2011.01.007. URL: https://www.sciencedirect.com/science/article/pii/S096663621100021X (visited on 06/30/2022).

[21] Luca Balvis et al. “Keep the Beat: Audio Guidance for Runner Training”. en. In: *Human-Centered and Error-Resilient Systems Development*. Ed. by Cristian Bogdan et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 246–257. ISBN: 978-3-319-44902-9. DOI: 10.1007/978-3-319-44902-9_16.

[22] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. en. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. ISBN: 978-0-262-03924-6.

[23] Chris M. Bishop. “Neural networks and their applications”. In: *Review of Scientific Instruments* 65.6 (June 1994). Publisher: American Institute of Physics, pp. 1803–1832. ISSN: 0034-6748. DOI: 10.1063/1.1144830. URL: https://aip.scitation.org/doi/abs/10.1063/1.1144830 (visited on 05/11/2022).

[24] *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 07/26/2022).

[25] Ronald J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. en. In: *Machine Learning* 8.3 (May 1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696 (visited on 05/11/2022).

[26] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: *The Annals of Mathematical Statistics* 22.3 (Sept. 1951). Publisher: Institute of Mathematical Statistics, pp. 400–407. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729586. URL: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-3/A-Stochastic-Approximation-Method/10.1214/aoms/1177729586.full (visited on 05/11/2022).

[27] Richard S Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 1999. URL: https://papers.nips.cc/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html (visited on 05/10/2022).

[28] Ivo Grondman et al. “A survey of actor-critic reinforcement learning: standard and natural policy gradients”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 42.6 (2012). Publisher: Institute of Electrical and Electronics Engineers, pp. 1291–1307. DOI: 10.1109/TSMCC.2012.2218595. URL: https://hal.archives-ouvertes.fr/hal-00756747 (visited on 07/05/2022).

[29] Qiang Liu et al. “Breaking the Curse of Horizon: Infinite-Horizon Off-Policy Estimation”. In: *arXiv:1810.12429 [cs, stat]* (Oct. 2018). arXiv: 1810.12429. URL: http://arxiv.org/abs/1810.12429 (visited on 05/10/2022).

[30] Benjamin Eysenbach and Sergey Levine. “Maximum Entropy RL (Provably) Solves Some Robust RL Problems”. In: *arXiv:2103.06257 [cs]* (May 2022). arXiv: 2103.06257. URL: http://arxiv.org/abs/2103.06257 (visited on 05/06/2022).

[31] Shangtong Zhang, Wendelin Boehmer, and Shimon Whiteson. “Generalized Off-Policy Actor-Critic”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/0e095e054ee94774d6a496099eb1cf6a-Abstract.html (visited on 06/11/2022).

[32] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *arXiv:1707.06347 [cs]* (Aug. 2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347 (visited on 05/11/2022).

[33] I. Csiszar. “I-Divergence Geometry of Probability Distributions and Minimization Problems”. In: *The Annals of Probability* 3.1 (Feb. 1975). Publisher: Institute of Mathematical Statistics, pp. 146–158. ISSN: 0091-1798, 2168-894X. DOI: 10.1214/aop/1176996454. URL: https://projecteuclid.org/journals/annals-of-probability/volume-3/issue-1/I-Divergence-Geometry-

of-Probability-Distributions-and-Minimization-Problems/10.1214/aop/1176996454.full (visited on 07/05/2022).

[34] Damien Ernst, Pierre Geurts, and Louis Wehenkel. “Tree-Based Batch Mode Reinforcement Learning”. en. In: *Journal of Machine Learning Research* 6.18 (2005), pp. 503–556. ISSN: 1533-7928. URL: http://jmlr.org/papers/v6/ernst05a.html (visited on 06/12/2022).

[35] Brett Daley and Christopher Amato. “Human-Level Control without Server-Grade Hardware”. In: *arXiv:2111.01264 [cs]* (Nov. 2021). arXiv: 2111.01264. URL: http://arxiv.org/abs/2111.01264 (visited on 05/05/2022).

A

Synchronous software architecture

In order to implement different kinds of deep reinforcement learning algorithms, a general architecture architecture is proposed in [35]. This implementation leverages a concurrent and synchronized execution framework designed for a better understanding of the code and a better use of the computational resources. A model of the proposed architecture can be seen in figure A.1.

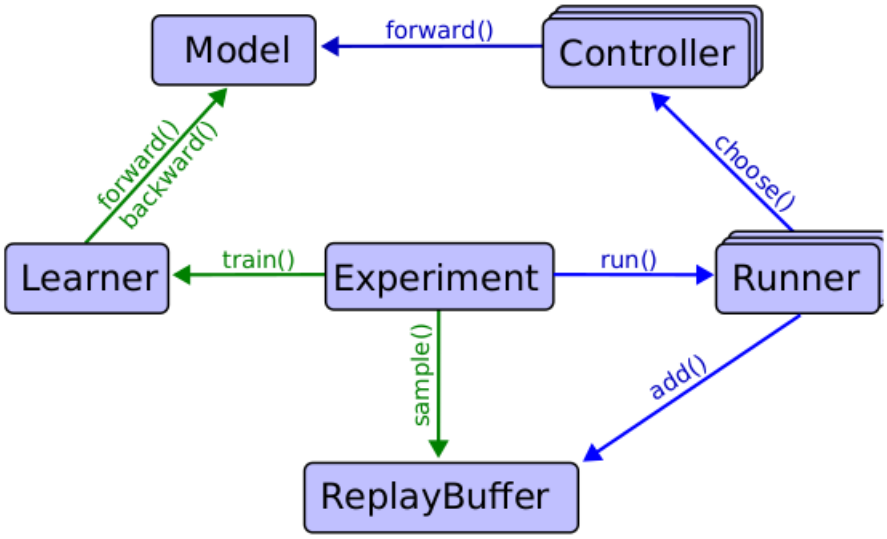

Figure A.1: Synchronous software architecture.

This architecture can be extensible for policy gradient algorithms and consist of the following elements:

- **Model**: Is usually implemented as a Neural Network. Return the probabilities distribution of the actions. For more complex approaches, many more models can be added for the value function and the IS estimator.
- **Controller**: Is the element that interprets the models outputs.
- **Runner**: Is the element that interacts with the environment, chooses its actions through the controller and add its experiences in the Replay buffer.
- **Learner**: Is the element that dictates how the model is trained. In this class is where the RL algorithms code are implemented.
- **Replay Buffer**: In this element we add all our experienced transitions and we sample from it.
- **Experiment**: Main class that allows to run the runner and sample batches of experiences for training the learner.