Ecole Polytechnique
Fédérale de Lausanne

Istituto Italiano
di Tecnologia

On the Benefits of Warm Starting for Sampling Model Predictive Control

A Reinforcement Learning Based Approach

Master Thesis

Author
SAVIOZ Baptiste

Professor - Lab
IJSPEERT Auke
Biorobotics Laboratory

Supervisor - Lab
SEMINI Claudio
Dynamic Legged Systems Laboratory

TURRISI Giulio
Dynamic Legged Systems Laboratory

Spring 2024

Abstract

In quadruped robotics, there are two main approaches for controlling a robot: The model-free approach, where a system representation is learned empirically, and the model-based approach, where the system is represented explicitly.

This project will present a framework that combines these two approaches, applied to quadrupedal-legged locomotion, using a sampling model predictive controller and a reinforcement learning policy.

The simplicity of the sampling MPC enables gait and foothold optimization, which is usually not possible with a standard MPC. The novelty of our approach is the warm start with an RL policy of the sampling MPC to increase performance.

The mitigated results underline the difficulty of reconciling these two fundamentally different approaches. Nevertheless, the results are encouraging, and the prospects for this approach are now more evident.

Contents

Abbreviation

MPC	Model Predictive Control
SRBD	Single Rigid Body Dynamics
CoM	Center of Mass
DoF	Degrees of Freedom
GRF	Ground Reaction Forces
PPO	Proximal Policy Optimization
LF	Left Front leg
RF	Right Front leg
LH	Left Hind leg
RH	Right Hind leg
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
ELU	Exponential Linear Unit
DLS	Dynamic Legged Systems Laboratory
GPU	Graphical Processing Unit
CPU	Central Processing Unit
SGD	Stochastic Gradient Descent
RQ	Research Question

Notation

Variables will be introduced throughout this work, which may have different significations with the context. If not specifically stated otherwise, the variable will always refer to the signification presented below:

f	Leg step frequency
d	Leg duty cycle
p	Leg touch down position
F	Foot ground reaction force
c	Contact status of the foot with the ground
J	Jacobian linking the foot to the base of the robot
s	State of the system
a	Action applied to the system
$\mathcal{W}$	Fixed world inertial reference frame
$\mathcal{B}$	Base frame attached at the robot's base

1 Introduction

In modern robotics, two primary approaches compete to determine how to control robots effectively: the model-based approach, which explicitly represents the dynamics and the environment, and the model-free approach, which relies on data-driven methods. With the emergence of ever more powerful simulation tools and learning algorithms, the model-free approach is gaining momentum. However, despite its recent progress, it still suffers from its opacity and lacks any guarantee of stability.

In this project, we will focus more specifically on the problem of locomotion in quadrupedal robotics, where neither side seems to have gained the upper hand yet. We propose a method that combines both approaches, using a Reinforcement Learning (RL) policy alongside a sampling Model Predictive Controller (MPC) to hopefully leverage the strengths of each approach.

A bit of history In their seminal work, Tassa et al. proposed a simple interface to democratize access to model-based control tools in robotics, specifically optimal control, such as model predictive control [Howell et al. 2022]. They studied and compared the best-performing control algorithms. To compare these results against a baseline, they also studied the model predictive sampling controller. To their surprise, what was intended as a benchmark for quantifying improvements performed surprisingly well.

Motivated by these results, Turrisi et al. studied the sampling MPC more specifically for quadrupedal robotic locomotion [Turrisi et al. 2024]. They showed that this controller greatly benefited from the increasing parallel computation capabilities. In addition, it could also easily optimize aspects of locomotion that were not usually possible with conventional model predictive controllers, such as gait. This project is being carried out within this context.

Sampling Controller Before presenting the objectives of this project, we need to describe the sampling controller briefly. Starting from an initial solution, known as a warm start, it will generate samples of actions, evaluate them on a model, and select the best one, making it the simplest form of optimizer. No information from a derivative is used, and the dynamics can be highly nonlinear, hence presenting multiple local minima. Therefore, the quality of the sampling controller solution relies extensively on the number of samples drawn and the quality of the warm start. Many algorithms have been developed to improve its performance, based on the nature of the problem [Kusumoto et al. 2019] or on stochastic criteria [Asmar et al. 2023][Pinneri et al. 2020].

However, this project takes a different approach. Instead of refining the convergence using conventional methods, the objective is to warm start the sampling controller in a better way with a solution generated by a policy obtained through reinforcement learning. This approach aims to combine the strengths of both model-based and model-free methods.

1.1 RQ1 : Is RL Warm Starting for the Sampling Model Predictive Controller Beneficial ?

The warm start is crucial for the convergence of the optimization for the controller. Indeed, the further the initial solution is from the optimum, the more it will be necessary to explore the search space, increasing the time and quality of convergence. In addition, if exploration is favored, it is at the cost of exploitation, and the final solution will lack precision. The intuition behind this Research Question (RQ) is to obtain the best possible warm start, avoiding the problem of exploration and maximizing the exploitation of the solution. This project will investigate whether a reinforcement learning policy can provide an initial solution that can benefit the sampling controller.

1.2 RQ2 : Can Multiple Warm Starting Be Beneficial for the Sampling Model Predictive Controller?

The intuition behind this second research question is that the problem fundamentally has several local optima, which can become more or less dominant depending on the system's evolution. For example, a trot and a crawl can be optimal successively, depending on the conditions. Obtaining an RL policy that is particularly well-suited to an optimum is relatively straightforward. However, training a policy capable of generalizing to several different optima and switching from one to another as the system evolves is a

much more difficult problem.
This is where the sampling controller can be helpful. Based on a reward function, it can evaluate different trajectories from various policies and select the best one. This allows two policies, each trained for different sub-problems with different optima, to be easily combined. The sampling controller could quickly switch from one optimum to another without waiting for its solution to converge. By combining several policies with different behaviors, we hope to address the exploration problem and improve overall performance.

State-of-the-Art With this in mind, an overview of the state-of-the-art techniques already used to combine an RL policy with an MPC will be given. First of all, RL policies can be used to optimize MPC parameters online [Zanon et al. 2021][Gros et al. 2020] or its constraints [Pandala et al. 2022][Martinsen et al. 2022]. These methods give excellent results and benefit from a solid guarantee of stability. However, they can only optimize what can be modeled with continuous and differentiable, as their optimizer is gradient-based. As a result, optimizing robot gaits and footholds based on discrete contacts is not possible, significantly reducing the movement diversity they can produce.

Another approach is to separate the problems. On the one hand, an MPC that optimizes ground reaction forces, and on the other, an RL policy that provides gait parameters [Yang et al. 2021] or footholds [Omar et al. 2023]. This method no longer limits the robot's gaits, but the latter parameters are not optimized and do not benefit from the guarantees offered by MPC.

The third approach adopted is learning an optimal MPC warm start by an RL policy trained offline to predict its results[Lembono et al. 2020]. It improves the solution's convergence and mitigates the problem of local minima, which affects all the techniques mentioned so far. Nevertheless, the MPC parameters are fixed, and the gradient-based optimization has the same limitations as mentioned above.

Objectives This project will attempt to answer these two research questions applied to quadrupedal robotics. For this purpose, the robot *AlienGO*, developed by *Unitree*, will be used, and experiments in simulation will be carried out.

This research aims to develop a controller that makes the robot walk in the best possible way. More specifically, the aim is to enable the robot to :

- follow an omnidirectional speed command, i.e., move from front to back, from right to left, and turn from right to left.
- be robust to noise and measurement errors
- be robust to external disturbances.
- be capable of navigating through various types of terrain

Outline To fulfill the objectives and answer the two research questions, the following structure is adopted:

- **Chapter 2** presents the foundational control scheme based on previous work at DLS.
- **Chapter 3** describes the control architecture adopted and the contributions made to it.
- **Chapter 4** summarizes training of a reinforcement learning policy
- **Chapter 5** introduce the adaptations made to the RL policy to achieve sampling MPC warm starting
- **Chapter 6** report briefly the experimental setup of the project.
- **Chapter 7** discuss in depth the results obtained in this project
- **Chapter 8** conclude this project with the final discussion

Context This project is being carried out as part of the master's thesis of Baptiste Savioz, a student in Robotics at EPFL, supervised by Professor Auke Ijspeert of EPFL's Bio Robotics Laboratory. The project takes place at the Italian Institute of Technology (IIT) in the Dynamic Legged Systems laboratory (DLS) of Dr. Claudio Semini under the supervision of Dr. Giulio Turrisi. DLS specializes in the locomotion of quadrupeds, notably having developed the HyQ robot [Semini et al. 2011], and has extensive experience in model-based solutions.

2 Background

2.1 Model-Free and Model-Based approaches

Robotic locomotion is a challenging problem, with a high-dimensional control space and usually a highly dynamic environment. As mentioned in the introduction, two main approaches to this problem have stood out: model-free and model-based methods.

Model-based approaches are the traditional methods. A model of the system and its environment is crafted for a control law to rely on. Optimal control strategies with robustness guarantees can be used. Moreover, the model predicts future states of the system, enabling planning strategies. Nonetheless, performance relies on the model's accuracy, which requires expertise and high *on-line* computational resources [Rawlings et al. 2017].

Model-free approaches on the other hand, do not rely on a predefined model of the dynamics but rather learn it from interactions with the environment, allowing them to adapt to complex environments. However, these approaches are less transparent and lack mathematical guarantees for stability. A model-based approach may perform poorly with unmodeled dynamics, but a model-free approach may exhibit unexpected behaviors. Additionally, learning dynamics requires extensive data and high *off-line* computational resources [Sutton et al. 1998].

A consensus on the best approach has not been reached yet. Model-free methods have shown great progress in recent years and can outperform model-based methods, especially on highly dynamic tasks [Cheng et al. 2023]. However, model-based approaches continue to excel in tasks that require precise control [Mastalli et al. 2022], such as manipulation tasks. This project aims to combine the strengths of both approaches into a unified framework.

2.2 Model Based Control

In this chapter, the model-based structure of the framework will be presented. It is the foundation for this project and has been available from the beginning. The architecture presented here, and the different building blocks come from the expertise built over the years of IIT's *Dynamic Legged Systems Laboratory* (DLS). However, small adaptations to the problem formulation have been made to anticipate the integration of the model-free part of this project. The control architecture is illustrated in figure 1, and the different components will be discussed in the following chapters.

From the control scheme, three main elements are identified:

- **The high-level control loop** : This provides planning for the system, including feet trajectories, ground reaction forces, and type of gait.
- **The low-level control loop** : Responsible for tracking references and transforming them into joint torques.
- **The Simulator** : Simulates the dynamics and the environment, providing the system states.

Moreover, the control of the legs differs between the swing and stance phases. The motivation for this is that the nature of these two problems is diverse. On the one hand, the stance leg affects the base with the ground reaction forces. A control law based on foot position would be highly sensitive to small position errors, making the design and tuning difficult. Therefore, relying on forces rather than position seems more suited for the stance controller since, ultimately, that is what affects the base. On the other hand, for the leg in the swing phase, the force of the foot is of no interest in comparison with its position. Accurately tracking the position of an end-effector with force control is a difficult task, so a position controller seems more appropriate.

2.2.1 MPC

This component is crucial to the control scheme and is the primary focus of this project. The Model Predictive Controller (MPC) maintains system stability by optimizing the ground reaction forces. Among

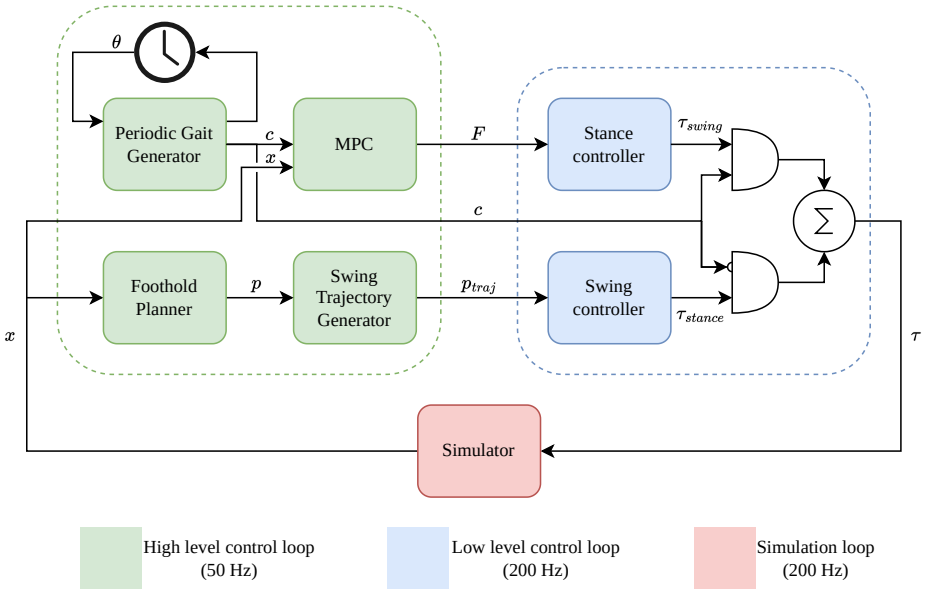


Figure 1: Model based control scheme

the various inputs to the system, ground reaction forces play the most critical role in maintaining stability. While the system can tolerate some errors in foot position, for example, it is highly sensitive to inaccuracies in force control. Therefore, these forces must be optimized *on-line* to ensure optimal performance. The optimal control problem to solve in its standard formulation is :

$$\begin{aligned} \boldsymbol{u}^*(x_0) = \min_{\boldsymbol{u}} \quad & \sum_{i=0}^{N} \left(x_i^T Q x_i + u_i^T R u_i\right) \\ \text{subject to :} \quad & x_{i+1} = f(x_i, u_i), \quad \forall i \in [0, N-1] \\ & x_i \in \chi, \qquad\quad \forall i \in [1, N] \\ & u_i \in \mathcal{U}, \qquad\quad \forall i \in [0, N] \end{aligned} \tag{2.2.1}$$

One will denote f as the model of the system, which is used to predict the future state. In this project, the Single Rigid Body Model (SRBM) is used. This is a simple model that assumes all the mass is concentrated in a point at the center of mass. It is a model commonly used in robotics [Orin et al. 2013] as it is computationally lightweight while still providing decent results. The exact model formulation is taken from previous work done at DLS [Rathod et al. 2021], and a complete derivation of the model is available in Appendix C.

Typically, a closed-form solution does not exist, and different approaches are available to solve this problem. Traditionally, it would be solved using a gradient- or Hessian-based solver [Frison et al. 2020][Gros et al. 2016]. However, in this project, a derivative-free optimizer, specifically a sampling-based optimizer, is used. It is the simplest type of optimizer, and its formulation is presented in Algorithm 1. Essentially, it evaluates M possible control actions and selects the best one.

Two significant advantages of this approach over other algorithms are its simplicity, avoiding the complexity of gradient-based optimizers, and its ability to perform the optimization process in parallel. Indeed, all the rollouts[1] can be simulated in parallel. As a result, the algorithm benefits considerably from GPU acceleration [Turrisi et al. 2024], enabling higher-rate planning. In their work on sampling-based MPC, [Howell et al. 2022] found that high-rate control is essential for maintaining stability and that a fast, approximate optimization often outperforms a more accurate solution achieved at a lower rate

Moreover, this sampling-based approach provides significant flexibility in problem formulation, as it does not require continuity and differentiability of the model. For instance, optimizing the discrete con-

[1] A rollout is a simulated trajectory of the system, given a model, a cost function, an initial state, and an input sequence

Algorithm 1 Sampling Controller

Require: cost function J, model f, warm start θ', variance σ, initial state x_0

for k = 1 to M **do** ▷ In parallel
  Sample $\theta_k \leftarrow \mathcal{N}(\theta', \sigma)$
  for i = 0 to N-1 **do** ▷ Iteratively
    $u_i = \pi(\theta_k, i)$
    $x_{i+1} = f(x_i, u_i)$
    $J_k = J_k + J(u_i, x_i)$
  end for
end for
Pick best sample $\theta^* \leftarrow argmin(J_k))$
return θ^*

tact sequence—determining when and where the robot's feet should make contact with the ground—can be easily incorporated into the same framework, which is typically not trivial with derivative-based optimizers, due for example, to the linear complementary constraints [Neunert et al. 2018]. These possibilities will be explored later in this project.

2.2.2 Gait Generator

The separation of the control between stance and swing phases requires the definition of a contact status c, which is the purpose of the gait generator. The evolution through time of the contact status defines the type of gait. An internal variable : the leg phase θ evolves with the leg frequency f. In the general case :

$$\theta(t) = \int_0^t f(t)dt, \qquad \in [0, 2\pi] \tag{2.2.2}$$

Then the contact status c is the comparison with the leg duty cycle d.

$$c(t) = \begin{cases} 1, & \theta(t) \geq d(t), \qquad \text{Stance phase} \\ 0, & \theta(t) > d(t), \qquad \text{Swing phase} \end{cases} \tag{2.2.3}$$

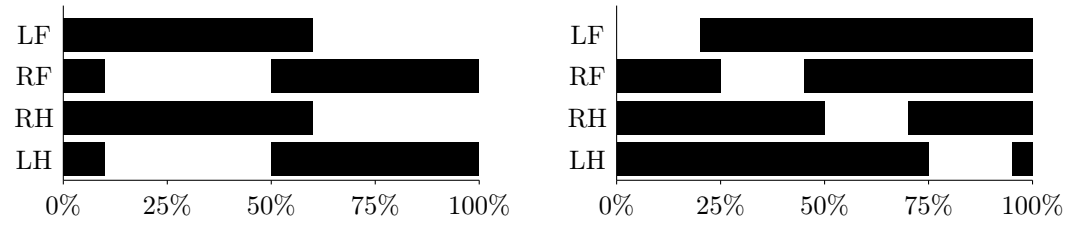
- f : The leg frequency $\in [0, 3][Hz]$
- d : The leg duty cycle $\in [0, 1]$
- θ : The leg phase $\in [0, 1[$
- c : The leg contact status $\in \{0, 1\}$

This formulation, using a single frequency and duty cycle per leg, provides flexibility and does not restrict the type of gait that can be generated. The different types of gait can be represented on Hildebrand-style diagrams [Hildebrand 1989]. At this stage of the project, the gaits are manually designed, with an example illustrated in Figure 2. The handcrafted gaits are usually periodic and predefined in the function of the task. A trot, for example, is more dynamic, while the crawl is the most static but often the most stable, with always at least three legs in stance.

2.2.3 Foothold Planner

The initial foothold planner is out of the scope of this project since it will be removed later. Implementation details can be found in previous work [Turrisi et al. 2024, p. 6]. Based on desired velocity and terrain elevation, it returns a desired touch-down position p_{TD} to be later tracked.

$$p_{TD,i} = p_{hip,i} + \frac{T_{stance}}{2} v_{ref} + \sqrt{\frac{z_{com}}{g}}(v - v_{ref}) \tag{2.2.4}$$



(a) Walking Trot (b) Crawl

Figure 2: Hildebrand-Style Diagrams of Quadrupedal Gaits, dark areas represent feet in contact with the ground throughout the gait cycle.

2.2.4 Swing Trajectory Generator

A swing trajectory defines the path of the foot during its swing or airborne phase. It is characterized by key parameters, including the lift-off and touch-down position and the step height. The swing trajectory generator aims to produce a smooth and controlled trajectory that mimics natural locomotion. Near touch-down and lift-off, it is preferable to maintain zero velocities to have smooth locomotion. On the one hand, to avoid discontinuity of the foot transitioning from swing to the stance phase, or vice-versa. On the other hand, large impact forces are minimized at touch-down, especially in the presence of uncertainties.

A swing trajectory controller will later track this trajectory and thus is defined in terms of position, velocity, and acceleration. The approach adopted in this project utilizes Bézier curves to mathematically describe the trajectory with heuristically defined control points [Zeng et al. 2019]. The properties of the Bézier curves allow the first and second derivatives to be computed, obtaining smooth velocity and acceleration. A comprehensive discussion of this method is provided in Appendix B. The parameters required by the swing trajectory generator are provided by a foothold planner, which can be based on heuristics or optimization:

- p_{LO} : The foot lift-off position
- p_h : The step height clearance
- p_{TD} : The foot desired touch-down position, simply referred to as p

2.2.5 Swing Controller

The swing controller tracks a swing trajectory for the foot in swing using a feedback linearization law [Alessandro 2011][Aude et al. 2022]. This formulation allows tracking a trajectory in task space (i.e., Cartesian space) instead of action space (i.e., joint space) using linear control techniques, PD-control in this case. The swing trajectory is defined for the foot position, velocity, and acceleration by the swing trajectory generator (appendix B).

The robot dynamic in joint space can be described as :

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) - J^T(q)F \tag{2.2.5}$$

The robot dynamic in Cartesian space can be described as :

$$\ddot{p} = J(q)\ddot{q} + \dot{J}(q)\dot{q} \tag{2.2.6}$$

With:

- τ : The joint torques $\in R^N$
- $q, \dot{q}, \ddot{q}$: The joint position, velocity and acceleration $\in R^N$
- $p, \dot{p}, \ddot{p}$: The end-effector position, velocity and acceleration $\in R^6$
- $M(q)$: The mass Matrix $\in R^{N\times N}$

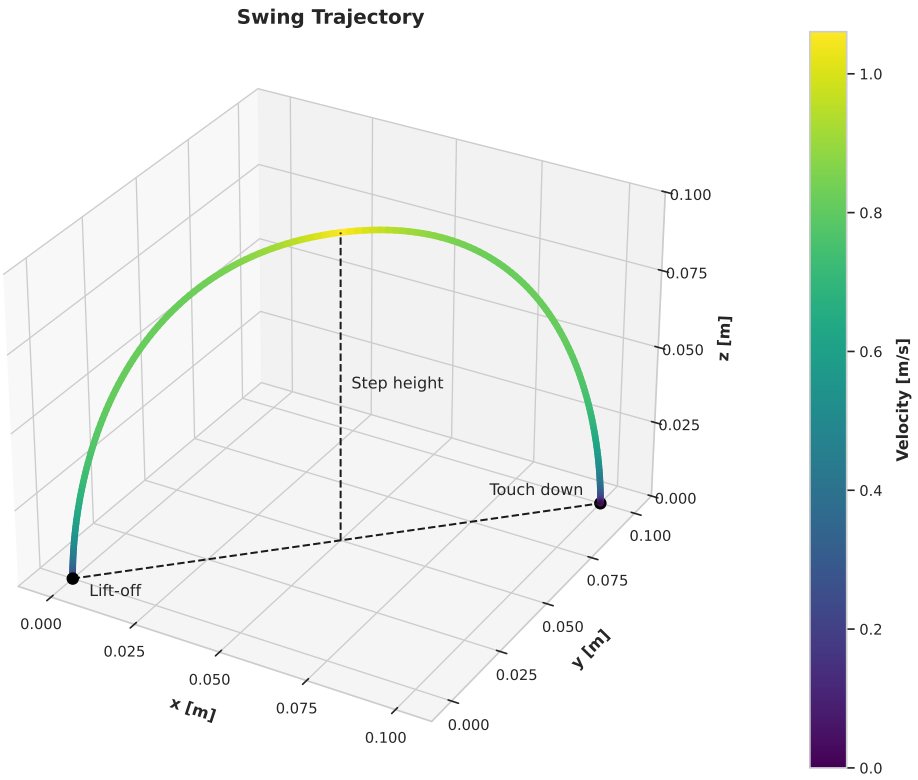


Figure 3: Example of a swing trajectory

- $C(q, \dot{q})$: The Coriolis and centrifugal forces $\in R^{N \times N}$
- $G(q)$: The generalised gravity in joint space $\in R^N$
- $J(q)$: The Jacobian in $\in R^{6 \times N}$
- F: The Generalised external forces $\in R^6$

As the leg is in swing, the influence of external forces can be safely neglected. Moreover, the robot leg is a 3-DoF kinematic chain; thus, the position and orientation cannot be jointly imposed. For this reason, the rotational dynamics are omitted, and the dimensions for p and J are reduced from 6 to 3. By manipulating (2.2.6) and (2.2.5), one can then relate the joint torques τ to the end-effector acceleration $\ddot{p}$:

$$\tau = M(q)J(q)^{-1}\left[\ddot{p} - \dot{J}(q)\dot{q}\right] + C(q, \dot{q})\dot{q} + G(q) \tag{2.2.7}$$

From equation (2.2.7), one can design a PD-controller in Cartesian space to track a reference trajectory $(\ddot{p}_{ref}, \dot{p}_{ref}, p_{ref})$ in the same space [Ding et al. 2021][Yang et al. 2021].

$$\ddot{p} = \ddot{p}_{ref} + K_D(\dot{p} - \dot{p}_{ref}) + K_P(p - p_{ref}) \tag{2.2.8}$$

With K_P and K_D, the position and velocity feedback gain.

2.2.6 Stance Controller

The stance controller presented in this project is not strictly a controller, as it does not involve neither tracking nor optimization. Instead, this controller translates commands for the stance foot from the high-level control loop space to the action space of the robot. In this context, the commands for the stance foot are ground reaction forces (GRF) F, while the action space commands are joint torques τ. The technique used is referred to as the *transpose Jacobian* technique, widely used in robotics for quasi-static motions [Yang et al. 2021], with J the end-effector Jacobian, q the joint position, and p the end-effector position.

$$\tau = J^T(q)F \tag{2.2.9}$$

This relation can be derived from the forward kinematics f and its derivative :

$$\begin{aligned} p &= f(q) \\ \dot{p} &= J(q)\dot{q} \end{aligned} \tag{2.2.10}$$

From the energy conservation principle, work done by the forces at the end-effector must match the work done by the joint torques. It can be expressed in the infinitesimal case (power conservation) as :

$$F^T\dot{p} = \tau^T\dot{q} \tag{2.2.11}$$

Finally, the relation (2.2.9) can be derived by substituting equations (2.2.10) and (2.2.11)[2]. This relationship is valid for generalized forces (force and torque) at the end-effector; however, only forces are considered in this project. Moreover, the force applied by the robot's leg is of the opposite sign with respect to the ground reaction force, which explains the negative sign in the implementation.

2.3 Possible Improvement

This framework has proven to be robust and efficient [Ding et al. 2021]. Nevertheless, there is still room for improvement. On the one hand, it relies on manually designed gaits that are all periodical, thus restricting the diversity of possible motion. On the other hand, the foot contact position is also based on a heuristic, and its potential is not fully exploited. Lastly, the quality of the solution depends on the number of samples and their randomicity, giving to the method a non-always deterministic behavior. In the course of this project, we will try to address these three areas of improvement using a reinforcement learning approach.

In other words, the reinforcement learning policy will try to provide gait parameters f and d and thus be able to craft any type of gait, from a simple trot to any aperiodic and asymmetric gait. In addition, it will also provide a foot touch-down position p, replacing the foothold planner. The approach aims also to reduce the variance of the solution and, thus, the non-deterministic behaviors.

[2]The demonstration has been slightly permissive with respect to the notation and makes the static case assumption. A more rigorous explanation can be found in *Joint Torques and the Jacobian Transpose* from [Caron 2023], and *Fast Numerical Methods for Inverse Kinematics* from [Bill 2002

3 Closing the loop : Blending RL warm starting into the sampling controller

To simplify the reader's comprehension, the final control architecture will be presented first, assuming that everything else works. This approach allows the reader to have a more global view and understand the choices made and their motivations. It does not follow the chronological order of development but rather reflects the order of thought.

It will be assumed that all the elements are in place to close the loop. First, policies capable of navigating through the imposed tasks were trained by reinforcement learning. Then, the problem was adapted so that the policies could predict actions compatible with the sampling controller's constraints. Finally, it is time to use these policies to warm start the sampling controller.

3.1 Control Scheme

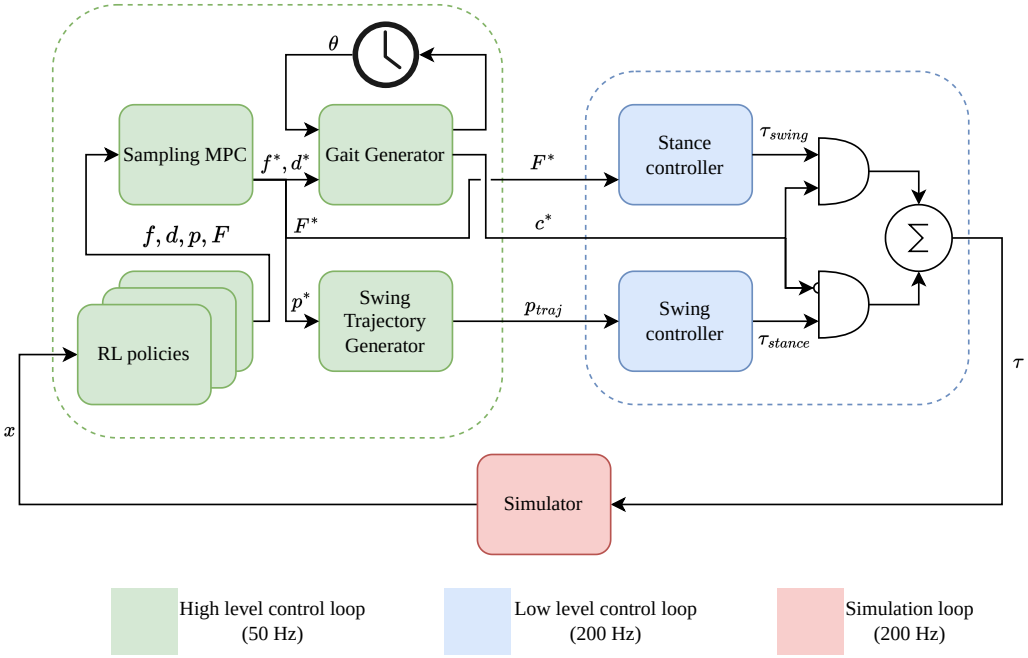


Figure 4: Control scheme with RL policy warm starting sampling MPC

3.2 The Sampling Controller

The sampling MPC formulation provides a simple way of optimising leg frequencies f, duty cycle d, feet touch down position p and ground reaction forces F from the RL policy solution.The following analysis examines how the different algorithm parameters affect performance.

3.2.1 Cost Function

Starting with the fundamentals, the cost function J, which is common for all types of MPC. In this project, the exact same cost function was used as in the work on the sampling controller carried out at the DLS laboratory [Turrisi et al. 2024]. It is a classic quadratic cost function, but with weights that have already been tuned for both this problem and the *AlienGO* robot, as presented in tables 9 and 10.

$$J(x_i, u_i) = x_i^T Q x_i + u_i^T R u_i \tag{3.2.1}$$

The state x and input u are the ones defined in the SRBD, detailed in Appendix C. In particular, the state x contains the Center of Mass (CoM) position, orientation, velocity, and angular velocity, as well as the four feet positions. The input u is composed of the desired feet touch down position p and the ground reaction forces F.

Algorithm 2 Sampling Model Predictive Controller with RL warm start

Require: cost func. J, model f, gait gen. g, warm start $\{(f,d,p,F)_{RL},...\}$, variance σ, init. state x_0

for k = 1 to M **in Parallel do**
  $(f,d,p,F) \leftarrow \{(f,d,p,F)^*,(f,d,p,F)_{RL},...\}$ ▷ Pick a sampling law
  $(f,d,p,F)_k \leftarrow \mathcal{N}((f,d,p,F),\sigma)$ ▷ Sample
  $c_{0\rightarrow N-1} \leftarrow g(f_k,d_k)$ ▷ Generate contact sequence
  for i = 0 to N-1 **Iteratively do**
    $p_i, F_i = \pi(p_k, F_k, i)$ ▷ Decode the actions
    $x_{i+1} = f(x_i, c_i, p_i, F_i)$ ▷ Step the environment
    $J_k = J_k + J(u_i, x_i)$ ▷ Update the cost
  end for
end for
$(f,d,c,p,F)^* \leftarrow argmin(J_k))$ ▷ Pick best sample
$p_0^*, F_0^* = \pi(p^*, F^*, 0)$ ▷ Decode the actions
return $(f,d,c,p_0,F_0)^*$

3.2.2 MPC hyperparameters

The number of M samples positively influences the quality of the solution but at the cost of computing time. However, since the computations are parallelized on the GPU, in reality this has little influence a plateau is reached, after which the computing time explodes. On the machine used in this project, the calculation time is almost constant up to 25'000 samples.

The model discretization time dt and the number of predictions N determine the prediction horizon H. A smaller dt results in more accurate model integration but shortens the horizon H. Conversely, increasing N extends the prediction horizon but lengthens the computation time. Therefore, a compromise needs to be found in order to have an accurate simulation and a sufficiently long horizon.

The influence of these hyperparameters on the performance of MPC sampling is analyzed and discussed in chapters 7.3.3 and 7.3.4. In this project, the following values will be used : $dt = 0.02[s]$, $N = 10$ for a horizon of $H = 0.2[s]$ and $M = 10'000$ samples.

3.2.3 Sampling Law

One can imagine various distributions from which to sample, the most obvious being a normal distribution, a uniform distribution, or any combination of both. In this project, both options were explored before opting for a normal distribution.

3.2.4 Sampling Variance

The variance determines the region around the mean in which the samples will lie. This concept is referred to as *exploration*. A higher variance allows for greater exploration, enabling samples to cover a larger space and increasing the likelihood of escaping local minima, leading to faster convergence. On the other hand, higher variance also means the samples are more diluted in space, resulting in a less precise approximation of the optimum, a concept known as *exploitation*. The choice of variance level is, therefore, a compromise between exploitation and exploration.

In this project, the following values will be used: $\sigma_f = 0.02[Hz]$, $\sigma_d = 0.05[\frac{rad}{2\pi}]$, $\sigma_p = 0.02[cm]$ and $\sigma_F = 10[N]$. They were tuned following an empirical approach.

3.2.5 Optimizing Different Variable

In addition, this framework provides an easy way of enabling or disabling the optimization of any of the variables f, d, p and F by setting their variance *sigma* to zero. This feature enables the measurement and comparison of their importance and performance in the optimization.

3.2.6 Multiple warm staring

Furthermore, combining multiple policies Π together is straightforward, by simply constructing a set of warm starts $\{(f,d,p,F)_i\}, \quad i \in \Pi$ and generating the samples depending on the desired proportions of the initial solutions. Alternatively, the previous solution can be used as a warm start, which will be referred to as *naive warm starting*.

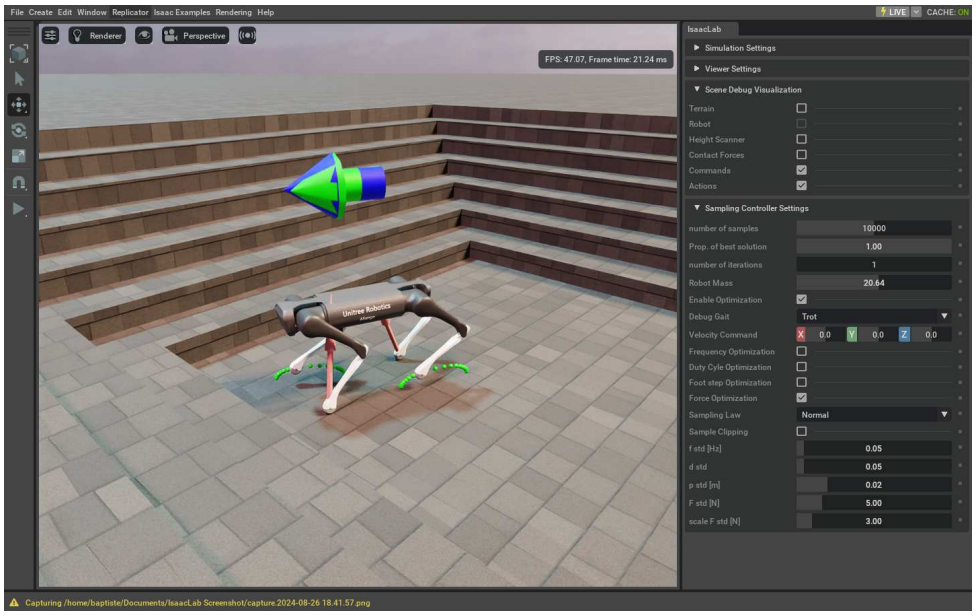


Figure 5: IsaacSim simulator, with on the right the panel developed to change the sampling controller parameters in real-time.

3.3 Summary

Finally, all the elements that make up the sampling controller have been reviewed, highlighting that it requires many decisions and adjustments, most of them empirical. To facilitate tuning, a visualization tool has been developed in IsaacSim, allowing various parameters to be modified in real-time. Similarly, another tool has been developed to visualize the performance of the sampling controller in real-time.

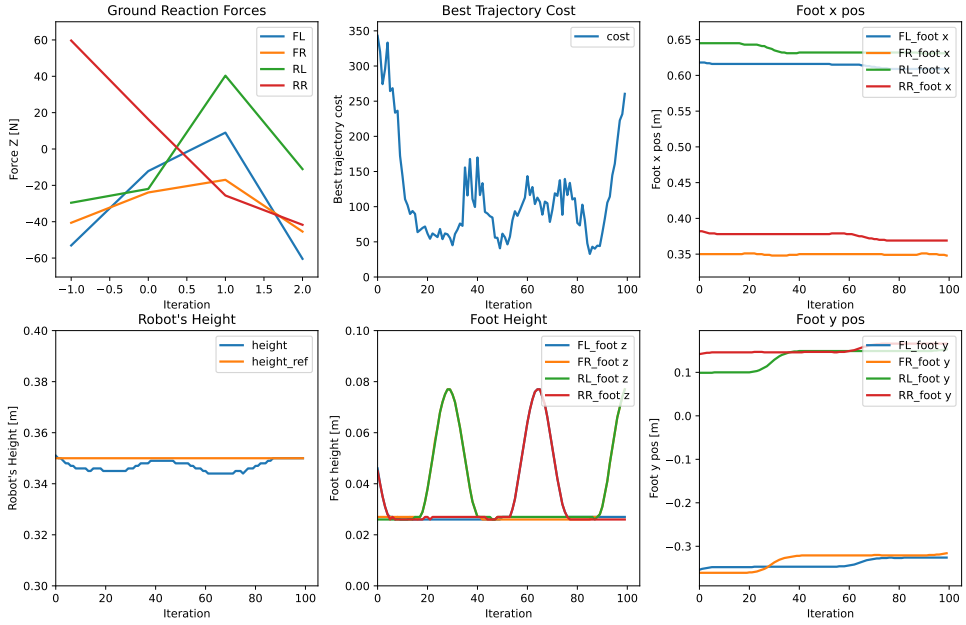


Figure 6: Live display of metrics of the sampling controller performances

4 Training a Reinforcement Learning Policy

This chapter will present the approach adopted in this project to train a reinforcement learning policy. Firstly, the theoretical aspects of Reinforcement Learning and the algorithm used will be introduced (chapter 4.1). Then the problem to be solved will be defined, along with the control scheme, the objectives and task of the policy, as well as its input and output space (chapter 4.2). Finally, the training of the policy with the various rewards and adaptations will be explained (chapter 4.3).

4.1 Reinforcement Learning

Reinforcement Learning (RL) involves training an agent to make decisions through trial and error, aiming to maximize rewards from its actions. It is particularly suited for complex tasks such as locomotion. This chapter covers key components of RL, starting with the Markov Decision Process, which lays the theoretical groundwork. It then moves to practical tools and frameworks that support RL implementations and concludes with a discussion of advanced algorithms that will allow the completion of complex locomotion tasks.

4.1.1 Markov Decision Process

A Markov Chain is a mathematical system that transitions between states based on a given transition probability. A Markov Decision Process (MDP) extends the Markov Chain by incorporating actions and rewards into the framework, making it suitable for modeling decision-making problems where an agent interacts with an environment. The key feature of an MDP, known as the Markov property, is that the future state depends only on the current state and action, with no dependency on the history of previous states. The MDP architecture is presented in Figure 7.

In robotics and machine learning, Reinforcement Learning (RL) frameworks rely heavily on MDPs to model the interaction between an agent and its environment. As a result, a user standard has emerged, and most RL frameworks and tools tend to align themselves with the *Gymnasium* interface [Towers et al. 2023], which provides a flexible MDP formulation capable of handling various types of problems. The framework used in this project follows this standard.

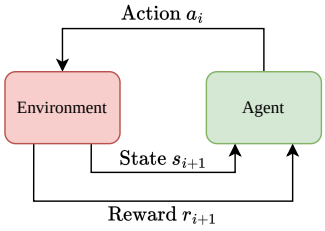


Figure 7: Markov Decision Process Diagram

4.1.2 Learning Framework and Simulator

The learning framework used in this project is called *IsaacLab*[3] [Mittal et al. 2023]. *IsaacLab* aligns with the *Gymnasium* interface, allowing it to leverage various tools developed within this ecosystem. Developed by *NVIDIA* in cooperation with *ETH*, *IsaacLab* is built on top of the powerful simulator *IsaacSim*, providing a robust combination for testing and development. A discussion on the choice of simulator can be found in chapter 6.1.1.

The different aspects of the problem are modularized and managed by managers, each responsible for specific functions within the Markov Decision Process, as illustrated in Figure 8.

- **Scene manager :** Manages the configuration of the simulation environment, such as setting physical parameters like gravity and friction. It also positions and defines elements like robots and sensors and designs the terrain to match specific test scenarios.

[3] *IsaacLab* was originally named *Orbit* and was renamed during the course of this research

- **Observation manager :** Defines the observation space of the policy and processes network inputs. This includes adding noise or transforming inputs to simulate real-world uncertainties or to better suit network architectures.
- **Reward manager :** Computes reward terms based on the outcomes of simulation steps to provide feedback to the agent.
- **Action manager :** Specifies the action space available to the policy and performs post-processing on network outputs. It also handles lower control schemes.
- **Event manager :** Schedules and executes actions that affect the simulation or other managers, such as altering terrain or adjusting reward structures. Events can be triggered at specific times.
- **Command manager :** Allows for the specification of user-defined commands that guide the agent's tasks, such as completing objectives or reaching specific points.
- **Termination manager :** Establishes criteria for ending simulation episodes, which can include time limits, robot malfunctions, or the completion of objectives.
- **Curriculum manager :** Adjusts the difficulty of the simulation based on the performance of the agent by modifying simulation or manager parameters.

This *manager* structure modularizes the problem and facilitates its implementation. The user can focus on the relevant implementations while re-using other more standard elements. This modularisation greatly accelerated the project.

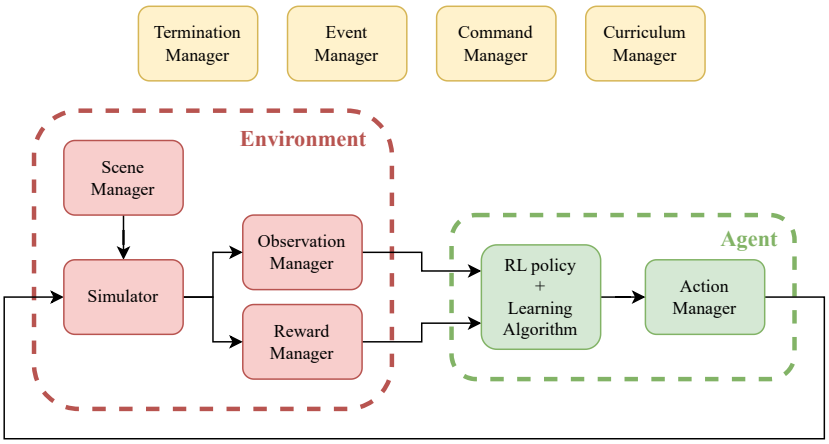


Figure 8: IsaacLab manager structure viewed as an MDP

4.1.3 Learning Algorithm

In the context of machine learning, a mathematical object is defined with parameters that are learnable. The objective is to learn parameters that produce the best outcomes to solve a problem. The mathematical object is commonly referred to as a *network* or a *model*. Reinforcement Learning is a field where the best outcome is unknown, and the quality of the outcomes must be evaluated using a so-called *reward function*. The reward is the only feedback available to the agent, and thus, learning optimal behaviors goes through trial and error. Specific algorithms have been developed for this purpose, and in recent years, trust region algorithms have gained prominence for their robust performance in complex environments, such as those encountered in robotic locomotion [Schulman et al. 2017b][Haarnoja et al. 2018][Fujimoto et al. 2018].

This project will use Proximal Policy Optimization (PPO)[Schulman et al. 2017a], a trust region algorithm that has become the most popular in Reinforcement Learning applied to robotic locomotion. Only the intuition behind PPO will be presented, and for the mathematical details, the reader is encouraged to refer to the original paper.

Proximal Policy Optimization aims to optimize an objective function $\mathcal{L}$, which relies on an advantage function A_t that quantifies how much the current action is better or worse compared to all the different possible actions. However, the advantage function is not directly observable, and therefore, an estimate

$\hat{A}_t$ must be employed [Latypov 2019]. Maximizing the objective function $\mathcal{L}$ relies on the assumption that the estimate $\hat{A}t$ is accurate. This is why the concept of a trust region is introduced: the policy update step must remain within a region where the estimate $\hat{A}_t$ remains reliable. PPO uses a surrogate[4] objective function $\mathcal{L}^{clip}$ to ensure the trust region, with a simple formulation that is yet effective compared to other types of trust region algorithms.

PPO uses an Actor-Critic structure, which is composed of two networks trained jointly. The Actor is responsible for predicting the agent's actions a_t and is trained with the surrogate clipped loss function $\mathcal{L}^{clip}$. In contrast, the Critic is responsible for predicting the advantage estimate $\hat{A}_t$.

The Actor-Critic structure enables a more stable and accurate estimate of future rewards and thus the estimated advantage $\hat{A}_t$. This dual structure not only refines the learning process but also stabilizes it by separating the evaluation of policy from the generation of action-value estimates. The structure of the algorithm is illustrated in figure 9.

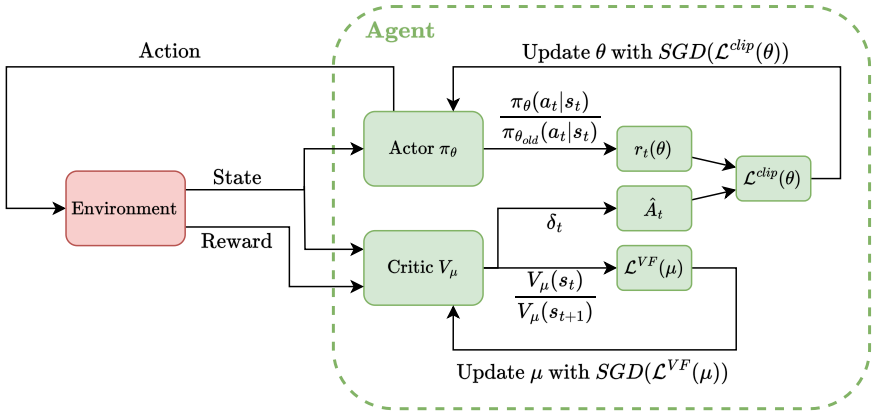


Figure 9: Markov Decision Process with PPO algorithm

4.2 Problem definition

4.2.1 Control Architecture

The control architecture defined in figure 1 is divided into a high-level, low-frequency control loop and a low-level, high-frequency control loop. The only parameters required by the low-level control loop to operate are feet ground reaction forces F, swing feet trajectories p_{traj} and a contact status c. At this stage of the project, the goal of using an RL approach is to substitute the high-level control loop.

The new control architecture is illustrated in figure 10. The heuristic that configured the *periodic gait generator* and *foothold planner* are replaced by the RL policy. Moreover, the MPC that optimized the ground reaction forces is not needed anymore, as they can be optimized by the network.

4.2.2 Action Space

The action space of the RL policy is the space that contains the policy's outputs, formally defined as :

$$\begin{aligned} \pi : \mathcal{S} &\longrightarrow \mathcal{A} \\ s &\longmapsto a \end{aligned} \tag{4.2.1}$$

with the low-level control loop, the action space of the RL policy has a physical meaning. For this purpose, a latent space $\mathcal{Z}$ is defined. The latent space is composed of :

- the gait parameters f and d, respectively the leg frequency and duty cycle that describe the locomotion gait, as described in chapter 2.2.2.

[4] *Surrogate* : A thing that substitutes for something else. In this context, the clipped loss $\mathcal{L}^{clip}$ is the surrogate of the loss $\mathcal{L}$

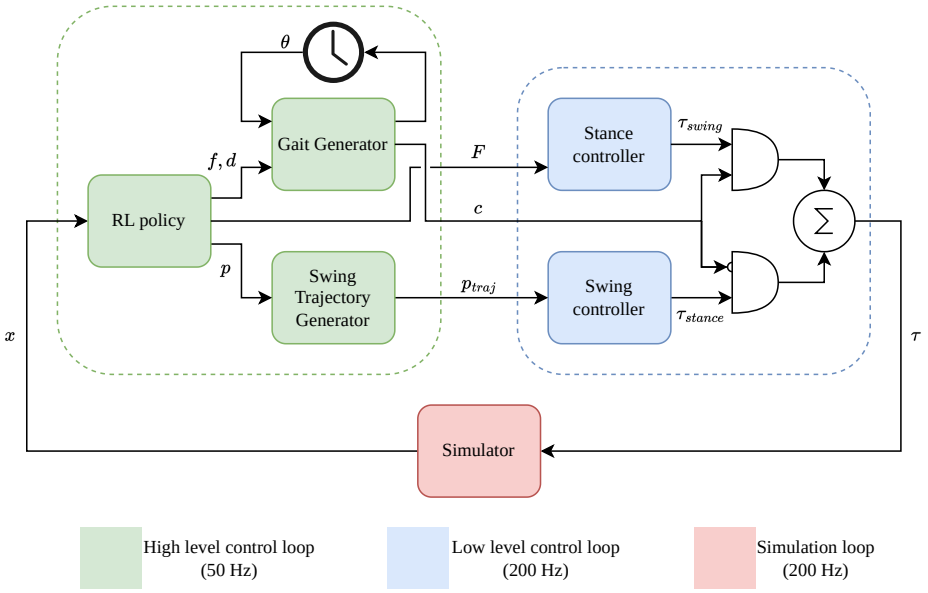


Figure 10: Control Scheme for training a Reinforcement Learning Policy

- The feet touch down position p, in horizontal frame $\mathcal{H}$.
- The feet ground reaction forces F, in horizontal frame $\mathcal{H}$.

It would be very difficult for the policy to learn a world frame reference, and thus a variable in the world frame $\mathcal{W}$. Instead, the actions are learned in the horizontal frame $\mathcal{H}$, which is a reference frame attached to the robot's base [Barasuol et al. 2013]. This frame is projected onto the world frame, maintaining the robot's yaw orientation while aligning with the horizontal plane.

In summary, the RL actions in the latent space $\mathcal{Z}$ are :

$$\pi(s) = (f, d, p, F) \in \mathbb{R}^{28} \tag{4.2.2}$$

Action Clipping

The outputs of the network are unbounded ($\in \mathcal{R}$), but the actions should be constrained to physically reasonable boundaries. There are two ways of ensuring that actions stay within a reasonable range: actions can be clipped, or deviations can be penalized.

The first method is simple and effective but tends to make learning more difficult, as the policy can receive identical feedback for different actions and can also be restrictive. The second method, on the other hand, facilitates learning while limiting the actions, in the manner of a barrier function in optimization problems but complexify the reward function.

In this project, we will use both approaches jointly. We will clip actions at their peak value and penalize them outside the sustainable regime. The clipping values for the various actions are shown in table 1. The leg frequency is clipped to maintain a feasible stepping frequency, while the duty cycle is constrained to its mathematical definition. The foot touch down position must lie within a slightly restrictive kinematically feasible region. Finally, the ground reaction forces are limited to feasible values and must remain within the friction cone, similar to the constraints used in the MPC (chapter 2.2.1).

4.2.3 Observation Space

The observation space $\mathcal{S}$ is the last frame that still needs to be defined. It should contains the right amount information for the network to be able to produce meaningful action.

Action	Clipping Range		
f	$\in [0, 3]$	Hz	
d	$\in [0, 1]$	$\frac{rad}{2\pi}$	
p_x	$\in [-10, +15]$	cm	around the hip
p_y	$\in [-10, +10]$	cm	around the hip
$\|\|F_{xy}\|\|$	$\in [0, \mu F_z]$	N	friction cone constraint
F_z	$\in [0, mg]$	N	

Table 1: Limit for the action clipping

The choice of reference frame is crucial, as it influences the network's ability to generalize its actions. For this reason, all the measurements of the robot's attitude are proprioceptive measurements in the base frame.

In addition to proprioceptive measurement, a height scanner is also available on *AlienGO*. This height scanner performed well in lab tests at *DLS*, so it was included in the observation space. The information on the terrain height is in the form of a discrete elevation map in the vicinity of the robot.

Finally, in Simulation, one has access to perfect measurements that will most likely not translate well to real-world hardware. To mitigate this issue, noise is added to the observations that will come from real-world sensors.

Quantity	Definition	Dimension	Noise
${}_{\mathcal{B}}v_c$	Center of Mass linear velocity	3	$\mathcal{U}(\pm 0.1)$
${}_{\mathcal{B}}\omega_c$	Center of Mass angular velocity	3	$\mathcal{U}(\pm 0.2)$
${}_{\mathcal{B}}g$	Projected gravity	3	$\mathcal{U}(\pm 0.05$
q	Joint position	12	$\mathcal{U}(\pm 0.01)$
$\dot{q}$	joint velocity	12	$\mathcal{U}(\pm 1.5)$
(f, d, p_0, F_0)	Last applied action in latent space $\mathcal{Z}$	28	
θ	Leg phase	4	
c	Leg contact status	4	
H	Discrete elevation map from height scanner	187	$\mathcal{U}(\pm 0.1)$

Table 2: Observation Space of the RL policy

4.2.4 Type of Network

The network architecture must be designed to be powerful enough to capture the complexities of the task yet as simple as possible to ensure efficient learning and prevent over-fitting.

For these reasons, the multi-layer perceptron (MLP) with Exponential Linear Unit (ELU) activation function has been selected. It is the architecture used in the PPO paper [Schulman et al. 2017a], widely adopted in robotics since then. In addition, pre-tuned networks architecture function of the robot and task complexity were available as baseline examples in *IsaacLab*.

The network used in this project is presented in figure 11, which benefited from these resources. It achieved great performance, making further tuning unnecessary. As presented before, PPO uses an actor-critic structure, which means that there are, in reality, two networks that share the same architecture.

4.2.5 Tasks

In order to achieve the stated objectives, which are to have the most robust omnidirectional locomotion possible across various types of terrain, four different tasks have been defined: base, rough, speed, and climb. Each of these tasks has different objectives and different problem configurations, which will be defined below.

The base task and rough task are general-purpose tasks that should be able to generate all types of movement. On the other hand, the speed task and climb task are specialized tasks that should excel, but

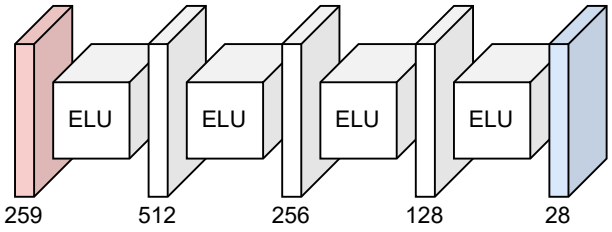


Figure 11: architecture of the *actor* and *critic* networks - 3 hidden layer MLP

only on one sub-problem.

The aim of these four tasks is to separate the different parts of the general problem into simpler sub-problems, with one policy trained per task. This, on the one hand, facilitates the tuning of the different policies and, on the other, generates movements and gaits that are both diversified and specialized. This is in the hope of combining them, thanks to the sampling MPC, into a single controller capable of doing everything.

Base Task This is the simplest task one can think of. The robot has to follow an omnidirectional speed command of up to $1[\frac{m}{s}]$, with no noise in the measurements and no external disturbances on perfectly flat terrain.

Rough Task The rough task, on the other hand, is the opposite of the base task. It has high noise in its measurements, as specified in table 2. It is also subject to strong external disturbances, notably: a force and torque on its base, a mass addition of $\pm 15\%$ of its weight, a variation in the terrain's coefficient of friction of $\pm 20\%$ and the robot is being pushed every few seconds.

In addition, the terrain is made up of various types of rough terrain, represented in figure 12, with varying degrees of difficulty. The rough task also requires omnidirectional speed control up to $0.8[\frac{m}{s}]$.

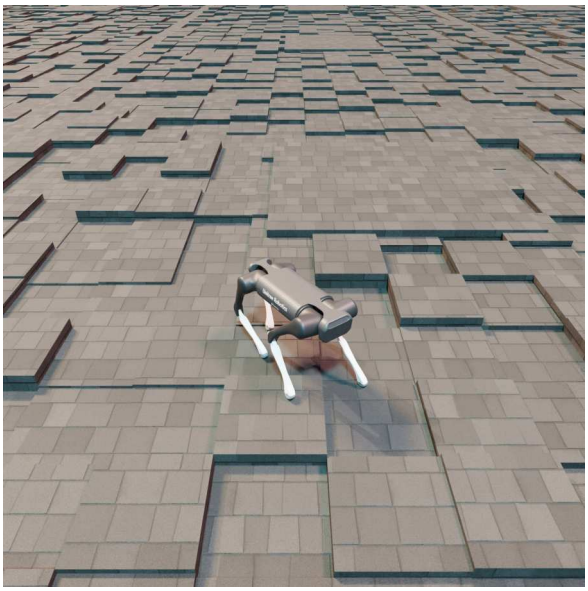
Speed Task The speed task is a specialized task, i.e. it answers only to a sub-problem. It must follow a command at the highest possible speed, up to $2.1[\frac{m}{s}]$, but mainly straight ahead, while minimizing its energy consumption. It has low noise in its observations, a third of the noise of the rough task, and light disturbances with a mass addition of $\pm 5\%$ and low force and torque on its base. the terrain is flat terrain, with some areas slightly rugged.

Climb Task The climb task is also a specialized task. It has to learn to climb up and down the stairs as high as possible. Its terrain is obviously composed of stairs, as shown in figure 12c, with different step heights, from 5 to 50 $[cm]$. Its speed command is low (up to $0.5[\frac{m}{s}]$) and essentially straight ahead.

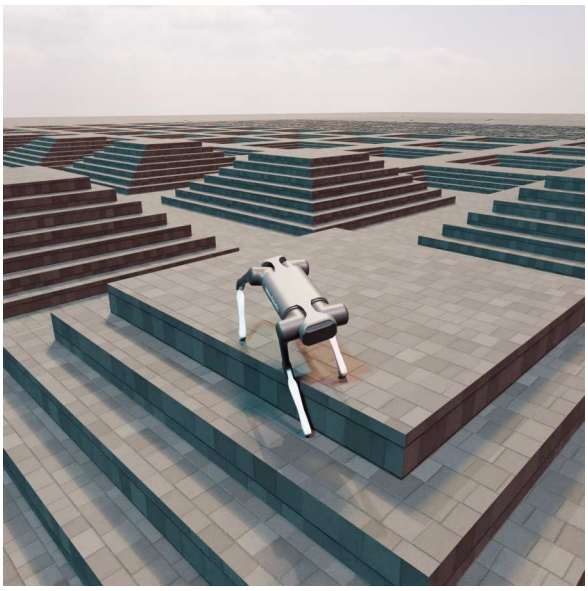
Task	Velocity Command		Terrain	Measurement Noise	Externeral Perturbation
Base	$\pm 1[\frac{m}{s}]$	omnidirectional	flat	no	no
Rough	$\pm 0.8[\frac{m}{s}]$	omnidirectional	all	high	high
Speed	$0 - 2.1[\frac{m}{s}]$	forward	flat&rugged	low	low
Climb	$0.3 - 0.5[\frac{m}{s}]$	forward	stair	low	low

Table 3: Configuration summary of the different tasks

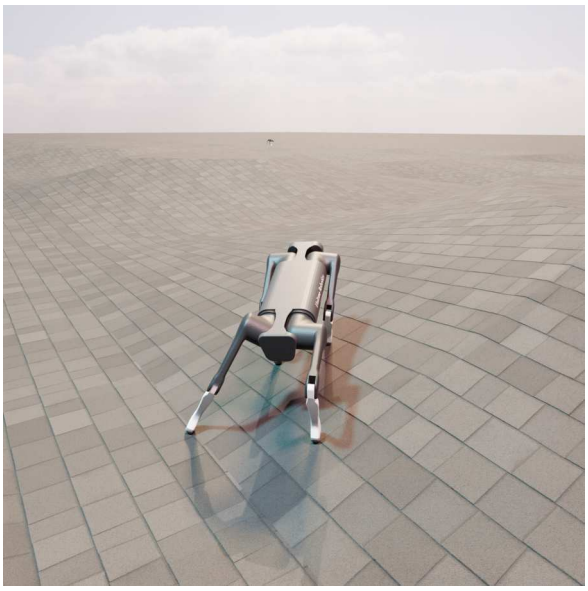
(a) Rugged terrain



(b) Box terrain



(c) Stair terrain



(d) Sloping terrain

Figure 12: Example of the robot *AlienGO* on the different terrains

4.3 Training

4.3.1 Reward

In Reinforcement Learning, the optimal action is unknown. This means there is a duality in the optimization problem. On the one hand, one needs to find the optimal model parameters for the network to best predict the optimal actions. On the other hand, the optimal actions are unknown, and one needs to find them. These optimization problems are optimized jointly in RL and it is made possible thanks to a reward function. It is the reward function that defines the optimality of the problem and thus needs to be carefully crafted. The reward function r is the weighted sum of individual reward term r_i rewarding or penalizing specific behaviors.

$$r(s_t, a_t) = \sum_i w_i \cdot r_i(s_t, a_t) \tag{4.3.1}$$

First, reward terms on the problem to be solved are obvious candidates, in this case, velocity tracking

reward functions. Then, terms on secondary features can also be safely added, like attitude tracking and energy consumption minimization. If the world was a place of mercy, this would likely be sufficient. However, these rewards are essential, but rarely sufficient to ensure a good policy, and especially a policy that will transfer well to the real world.

For these reasons, one needs to enter the world of reward shaping, where behaviors that are feasible and optimal only in simulation have to be artificially removed. For example, penalizing the action rate or avoiding the joint limit helps to avoid excessive jerk on the motor. One can also penalize sliding that often results from simulation exploitation.

All the reward terms used in this project will be listed with a short discussion in the following, and then their associated weight depending on the task will be listed in table 4.

Reward Term Function

Before listing the different terms, the functions commonly used to calculate rewards—sometimes referred to as *kernels*—will be presented[5].

- **Squared Exponential Kernel (SEK)** is a common function used for task rewards, given its bounded set. The tightness of tracking can be controlled with the hyperparameter σ.

$$r_i(x, x_{ref}, \sigma) = e^{\frac{(x-x_{ref})^2}{2\sigma^2}}, \quad r_i \in]0, 1] \tag{4.3.2}$$

- **Soft Squared Exponential Kernel (SSEK)** is an adaptation of the squared exponential kernel developed in this project to provide flexibility in certain directions, thanks to an additional hyperparameter α. The full mathematical development can be found in the Appendix.D.

$$r_i(x, x_{ref}, \sigma, \alpha) = softexp(x, x_{ref}, \sigma, \alpha), \quad r_i \in]0, 1] \tag{4.3.3}$$

- **Quadratic Penalty (L2)** is a term generally used for penalties. It is used to quadratically penalize a deviation. Its ensemble is not bounded.

$$r_i(x, x_ref) = (x - x_{ref}) \cdot (x - x_{ref}), \quad r_i \in \mathbb{R}_+ \tag{4.3.4}$$

- **Linear Penalty (L1)** is also used for penalties but penalizes large deviations less strongly. Its ensemble is not bounded.

$$r_i(x, x_{ref}) = x - x_{ref}, \quad r_i \in \mathbb{R}_+ \tag{4.3.5}$$

Primary Objectives

The primary goal of the robot is to follow a velocity command v_{ref}, defined in the base frame $\mathcal{B}$. This velocity is defined for three components: the velocity in x and y as well as the angular velocity in z: $(\dot{x}, \dot{y}, \dot{omega}_z)_{\mathcal{B}}$, which fully defines planar locomotion. In addition, the reference of the other three components $(\dot{z}, \dot{\omega}_x, \dot{\omega}_y)$ are set to zero to ensure stable robot attitude. Deviation from these references is more or less penalized based on the task. For example penalizing vertical speed z can restrict certain gaits like galloping, which naturally involve more vertical movement than trotting.

Furthermore, some tasks are not particularly suited to following a speed setpoint. For example, when navigating through complex terrain such as stairs, following a precise speed setpoint is less important than making progress through the terrain. The approach generally used to solve this kind of problem is waypoint navigation [Hoeller et al. 2023b][Cheng et al. 2023], but this approach is not compatible with the MPC sampling formulation used in this project. To address this, a new reward function has been developed: the *Soft Squared Exponential Kernel* (SSEK). This provides a reference velocity plateau rather than a single value to give the policy room to learn an optimal velocity profile and improve its performance. The size of this plateau is set by the α hyperparameter and is proportional to the difficulty of the terrain.

[5] In *IsaacLab*, these functions are referred to as *kernels*, although this usage is somewhat imprecise.

Secondary Objectives

A secondary objective is attitude tracking, which aims to keep the robot flat and stable. To achieve this, the flat orientation penalty is defined. It is calculated by penalizing the norm of the gravity vector projected onto the x and y components in the base frame $||(g_x, g_y)_{\mathcal{B}}||$. In addition, a height tracking reward is defined to keep the robot at a height reference. The height is calculated proprioceptively in the base frame $\mathcal{B}$. Note that the height is defined only if at least a foot is in stance.

$$h = z_{base} - \frac{\sum_i c_i z_{feet}}{\sum_i c_i} \tag{4.3.6}$$

In addition, to develop efficient locomotion, energy consumption must be penalized. To do this, one could simply penalize motor torque, which seems reasonable. However, there is a metric more suited to locomotion, dimensionless and comparable across robots of different morphologies and sizes: the Cost of Transport (CoT). Its energy consumption is normalized by speed and robot weight.

However, its computation in simulation is not trivial, as the energy consumed is unknown. It is possible to determine the mechanical power in a joint using the relationship $\tau\dot{q}$. However, this is a simplistic view of the energy consumed by the robot, as it neglects the motor's no-load consumption (stalk torque) and assumes that all deceleration energy can be recovered (perfect regenerative breaking). To address these limitations, the CoT formulation used in *fast and efficient locomotion via learned gait transition* from Yuxiang Yang et al. will be used [Yang et al. 2021].

$$CoT = \frac{\sum_{i=1}^{Q} max(\tau\dot{q} + \alpha\tau^2, 0)}{mgv} \tag{4.3.7}$$

Reward Shapping

The following describes the various techniques used to limit undesirable behavior. These undesirable behaviors most often result from the exploitation of the simulator. Indeed, as powerful as the simulation may be, it remains an imitation of reality that cannot capture all its details. For example, since motor overheating and jerk are not modeled, the policy is unable to minimize them. To limit this, it is common practice to penalize the action rate ($\dot{a}_t$) in order to limit the jerk and to penalize joints that are close to their mechanical limits (joint soft limit).

In addition, the actions are clipped, but to prevent the network from outputting values too far from the applied values, the difference between the applied action and the raw action is penalized (clipping violation: $a_{raw} - a_{clip}$). Since the friction cone constraint is imposed in the clipping of actions, this also penalizes slipping [6].

Finally, in some tasks, in order to impose behavior, actions are penalized for remaining in a space more restricted than the physical constraints (action constraint).

	Reward Term		Base Task	Rough Task	Speed Task	Climb Task
Primary objectives	$v_{xy,ref}$	SEK(σ=0.5)	1.5	1.5	1.5 [7]	1.5 [8]
	$\omega_{z,ref}$	SEK(σ=0.5)	0.75	0.75	0.75	0.75
	$v_{z,ref}$	L2	-2	-2	-2.5	-2
	$\omega_{xy,ref}$	L2	-0.05	-0.05	-0.2	-0.05
Secondary objectives	z_{ref}	SEK(σ=0.1)	0.2	0.2	0.1[9]	-
	flat orientation	L2	-1	-1	-	-
	CoT	L1	-0.04	-0.04	-0.05	-
Reward shaping	Action rate	L2	-2.5e-5	-2e-5	-2e-5	-2e-5
	Joint soft limit	L2	-3	-3	-	-1
	Clipping violation	L2	-	-0.1	-0.1	-0.1
	Action limit	L1	-	(-2,-1,-,-)	-	(-1,-,-,-)

Table 4: Summary of all the reward terms for the four different tasks

[6] Not entirely true, as with more than one leg in contact, it is possible to slip without violating the friction cone constraint
[7] Tighter reward with σ=0.4
[8] Soft squared exponential kernel SSEK, with $\alpha \in [0, 0.5]$, proportional to terrain's difficulty
[9] Soft squared exponential kernel SSEK, with α such as there is $\pm$1.5cm tolerance on the height

4.3.2 Noise and Randomization

In order to limit the sim-to-real transfer gap and obtain a policy that is both robust in simulation and the real world, randomizing the simulation parameters and perturbing the system have been demonstrated to be beneficial [Lee et al. 2020]. To achieve this, the event manager in *IsaacLab* was leveraged, and the following terms were defined:

- **Friction coefficient randomization** with $\mu_s \in [0.6, 0.8], \quad \mu_d \in [0.4, 0.6]$.
- **Base mass randomization** with $m \in [85\%, 115\%]$ of the robot mass.
- **External perturbation on base** with bias force $F \in [-10, 10]N$ and torque $\tau \in [-1, 1]Nm$.
- **Reset joint position randomization** with $q \in [0.8, 1.2]$ of the default joint position.
- **Robot push** by setting a base velocity $\in [-0.5, 0.5]\frac{m}{s}$ in a random direction every 8-12 seconds.

The values given here are for the *high* external perturbation listed in table 3. The *low* external perturbations are a third of these values.

4.3.3 Curriculum

In order to stabilize the training, curriculum learning was used, progressively increasing the difficulty of the task according to the performance of the robot. For the climb and rough task, it was the difficulty of the terrain that evolved, presenting increasingly uneven terrain and higher and higher stairs. For the speed task, it is the speed command that increases as the robot progresses. This approach enables the policy to gradually learn initial locomotion skills in a simple environment and then refine these skills on more demanding tasks.

Furthermore, to accelerate learning, the action rate and action limit penalties were activated only after the police had learned to walk. This increases the exploration of movements before optimizing them, effectively accelerating the learning process.

4.3.4 Normalisation

A crucial factor for successful training is the normalization of the network's outputs, as networks struggle with outputs that vary widely in scale. For example, in this project, ground reaction forces can reach hundreds of Newtons, whereas leg frequencies are generally limited to a few Hertz, underlining the importance of normalization.

However, standardizing the different actions is not straightforward. The approach adopted to find the parameters is based on empirical results and the study of locomotion parameters. To allow these parameters to generalize as well as possible, they are based on robot characteristics such as mass and hip height.

Action	shift	scale	unit
f	1.5	0.2	Hz
d	0.6	0.03	$\frac{rad}{2\pi}$
p	0	5% hip height	cm
F_{xy}	0	2.5 % mg	N
F_z	$\frac{mg}{\sum c_i}$	10% mg	N

Table 5: Normalizing parameters

This concludes the chapter *Training a Reinforcement Learning Policy*. The results are presented and discussed in chapter 7.1.

5 Distillation of an Export Policy into a Student Policy

The RL policies previously introduced have demonstrated strong performance in simulations. However, their current formulation is unsuitable for warm starting a sampling controller, which is ultimately the goal of this project. This chapter will first recall the formulation of the RL policies at this stage, followed by the formulation required for the sampling controller. Finally, different possibilities to conciliate the two will be presented and discussed.

5.1 The change in Action Space and Necessary Preprocessing

The RL policies π introduced in chapter 4 predict an action a_t based on an observation s_t. In other words, the policy provides a single output for each input.

However, the sampling controller K, a type of Model Predictive Control (MPC), predicts states and optimized actions over a horizon H given an initial state and actions for that horizon. In other words, the output space of the RL policy π is not directly translatable to the input space of the sampling controller K.

$$\begin{aligned} \pi &: \mathcal{S} \longrightarrow \mathcal{A} \\ s_0 &\longmapsto a_0 \end{aligned}$$

(a) RL policy domain definition

$$\begin{aligned} K &: (\mathcal{S}, \mathcal{A}^H) \longrightarrow (\mathcal{S}^{H-1}, \mathcal{A}^H) \\ (s_0, a_0, ..., a_{H-1}) &\longmapsto (s_1, ..., s_{H-1}, a_0^*, ..., a_{H-1}^*) \end{aligned}$$

(b) MPC domain definition

This change of space requires a few transformations before reinforcement learning can be used to warm start a sampling controller.

In addition, the effectiveness of the sampling controller is often inversely proportional to the size of the search space. By keeping the search space as compact as possible while ensuring sufficient action diversity, the controller can operate more effectively, focusing on a relevant subset of actions and refining its policy more efficiently. Encoding the actions provides an effective method to reduce the search space. For example, interpolation can reconstruct the original actions with minimal reconstruction error and fewer parameters. Previous work [Turrisi et al. 2024] and [Howell et al. 2022] has already addressed this problem and discussed the efficiency of different action encoding methods.

The following paragraphs will present and discuss different transformations and encoding methods.

Full Discrete Actions : The naive take on this problem of single input multiple outputs would be to have a policy that predicts a full discrete action for each time step of the horizon. However, this would result in a large search space. It is presented here for completeness but will not be used in this project.

$$\begin{aligned} Actions &= (f_t, d_t, p_t, F_t)_{i \in [0,H]} \\ Dim &= 28 \times H \end{aligned} \tag{5.1.1}$$

Other approaches will now be considered

Single Gait Parameters Set : First, the time horizon is relatively small compared to the periodicity of the gait. At most, two steps can be predicted. The influence of the variation of the gait parameters along the horizon is relatively minor. In addition, if reactive behavior is required for the robot to remain stable, it would increase the stepping frequency at the beginning of the horizon. For these reasons, considering a fixed frequency and duty cycle, taken as the first input, for the entire horizon duration seems a reasonable assumption. This is the first action encoding: a single gait frequency and duty cycle along the horizon.

$$\begin{aligned} Actions &= (f_0, d_0) + (p_t, F_t)_{i \in [0,H]} \\ Dim &= 8 + 20 \times H \end{aligned} \tag{5.1.2}$$

This is the only encoding considered in this project for the gait parameters. Its usage is now implicitly implied.

Single Touch Down : The touch-down position is tricky to optimize since not every provided input will be applied in the centroidal model, but only the one at touch-down. As a result, ensuring temporal consistency with the discrete action is challenging, making its optimization especially difficult. Similar to the reasoning for single gait parameters, the prediction horizon is relatively small compared to stepping. Considering a unique touch-down position prediction throughout the horizon also seems a reasonable assumption.

$$\begin{aligned} Actions &= (f_0, d_0) + (p_0) + (F_t)_{i\in[0,H]} \\ Dim &= 8 + 8 + 12 \times H \end{aligned} \tag{5.1.3}$$

Cubic Spline : The action encoding reduces the number of features while ideally maintaining a low mismatch error. This implies grasping the nature of the problem in order to find a meaningful encoding. A powerful parameterization is a spline, with the number of polynomials and the polynomial order as parameters. Prior work [Turrisi et al. 2024] and [Howell et al. 2022] experimented with linear, square, and cubic splines. Their results show the effectiveness of the cubic spline with a single polynomial, which seems natural since it can maintain continuity in the first two derivatives with the lowest number of parameters.

$$\begin{aligned} Actions &= (f_0, d_0) + (\theta_{p,i}, \theta_{F,i})_{i\in[0,3]} \\ Dim &= 8 + 4 \times (8 + 12) \end{aligned} \tag{5.1.4}$$

The mathematical derivation and a complete discussion of the cubic spline used in this project are available in Appendix A.

Finally, any combination between the presented encoding is also possible. The performances of the different encodings are presented and discussed in chapter 7.2.

5.2 Imitation Learning

The challenge of aligning the action space of RL policies with the input requirements of the sampling controller has been previously discussed, and different possible input spaces for the MPC have been presented. In the traditional RL framework, only the first action is applied. Hence, the network does not receive feedback on subsequent actions, impeding its ability to learn any of the presented MPC input spaces effectively. This chapter will present and discuss different approaches to solve this problem and enable us to learn a policy that can effectively warm start a sampling controller.

Two main approaches have been considered. The first approach is to record a trajectory of observation and action pairs from an RL policy, referred to as the *expert policy* from now on. From a trajectory, one can construct a datapoint with the first observation and by encoding the action trajectory of multiple single actions into one of the encodings presented above. By repeating this process, one could build a dataset on which a new *student policy* can be trained with the desired output space. This method is called *imitation learning*, a type of *supervised learning*.

The second approach involves training a policy directly with the desired output space. This method calculates rewards for unapplied actions using the MPC formulation to assess the cost of a *rollout*. This approach leverages the MPC cost function, aligning the policy closely with optimal control objectives. However, it has the drawback of a very large output space for a Reinforcement Learning problem, which is inherently more difficult to solve than a supervised learning problem. Despite this challenge, this approach has the potential to yield a policy that is more closely aligned with the final application, providing greater accuracy and efficiency in complex tasks. This chapter will focus on the imitation learning approach, and the second approach will be discussed later in chapter 5.3.

5.2.1 The Naive Implementation

The naive implementation of imitation learning provides a straightforward method to train a policy by directly mimicking an expert policy's actions. This approach involves generating a dataset from the expert's actions, followed by training a neural network to predict these actions using a supervised learning algorithm.

Generating the Data The first step in the naive implementation is generating a dataset that captures the relationship between observations and the expert policy's actions.

In general, the observation space of the expert and the student does not have to be the same. Often, the expert network has access to privileged information compared to the student. However, the action spaces of the two networks must normally be equivalent in order to train the student with the knowledge from the expert. In this chapter, the adaptation to make this possible will be presented.

Initially, a trajectory of observations and corresponding expert actions is collected over a horizon H. These expert actions must then be encoded into the desired output space for the student policy. The actions are encoded with a function P, which is trivial except for cubic spline encoding. Indeed, with H actions spaced in time, four spline parameters must be determined to reconstruct the actions best. This is an interpolation problem for which an exact closed-form solution exists. The details are presented in Appendix A.2.

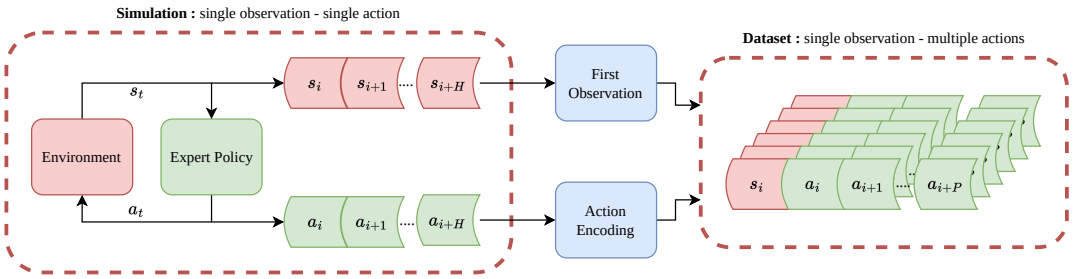


Figure 14: Dataset Generation

The Network The choices of networks are quite numerous in supervised learning. Complex networks, especially with some kind of temporal dependency, have been considered. However, the Multi-Layer Perceptron (MLP) of the expert policy demonstrated excellent results on the tasks defined in this project. In addition, it is a simple network that facilitates the implementation and optimization. Therefore, the same network architecture as the expert policy (actor-network) has been adopted for the student policy.

The Algorithm The supervised learning process involves selecting appropriate components such as the optimizer, loss metric, and learning rate scheduler. The objective is to minimize the data fitting error between the expert encoded actions and the actions predicted by the student network, making the Mean Squared Error (MSE) loss a natural choice. A critical implementation detail is that the data are scaled and transformed later in the *Action manager*, meaning that the network outputs are (mostly) normalized and unitless. This normalization ensures that the MSE loss remains pertinent across different dimensions of the output space. The Adadelta optimizer, known for its simplicity and effectiveness, has been chosen, along with a basic learning rate decay to ensure convergence.

The naive implementation of imitation learning offers a direct and accessible approach to training a student policy by leveraging the knowledge encapsulated in an expert policy, which would serve as a baseline for other methods. Although the naive *supervised learning* implementation is a valid approach to solving this problem, it does not benefit from any of the *imitation learning* lessons. The subsequent chapters will explore a method specifically designed to address these limitations, aiming to enhance student policy performances.

5.2.2 DAgger

The difficulty in imitation learning lies in the variety of observations. When a trajectory is collected using the expert's actions, those actions influence the next observation, and so on. This observation can be thought of as an *expert* observation. A student policy put in the same situation would predict an action with an error compared to the expert, resulting in a slightly different observation. After only a few iterations, the student's trajectory would have diverged significantly compared to the expert and would face observations never encountered in training. This concept is known as *regret*. The regret of the naive implementation is quadratic [Liu 2019], which makes training a good policy especially difficult. This issue can be overcome by long training on large datasets with great diversity (generated, for example, by adding noise or disturbances).

Algorithm 3 Naive Imitation Learning Algorithm

Require: Expert policy π_{exp}, action encoding P
Dataset Generation:
Initialize $\mathcal{D} \leftarrow \emptyset$
for $i = 1$ **to** M **do**
    Collect a trajectory : $\{(s_t, a_t)\}_{t\in[1,H]}$
    Encode the actions : $a_{enc} = P(a_1, a_2, ..., a_H)$
    Add the datapoint to the Dataset : $\mathcal{D} \leftarrow \mathcal{D} \cup (s_0, a_{enc})$
end for
Policy Training
Initialize π_0
for $i = 0$ **to** N **do**
    Train policy π_{i+1} : $\pi_{i+1} \leftarrow \text{Train}(\pi_i, \mathcal{D})$
end for
return π_N

Nonetheless, dedicated algorithms with reduced regret have been developed. This chapter will present the Dataset Aggregation algorithm, DAgger, introduced by [Ross et al. 2010], which leverages online expert querying to achieve no regret.

How DAgger Works The DAgger algorithm utilizes the same training algorithm, network architecture, and action encoding as the naive implementation, with the primary difference being how the training dataset is generated.

The key idea behind DAgger is to iteratively query the expert policy not only on the expert's trajectory but also on the student policy's trajectory. The process begins by allowing the student policy to generate a trajectory based on its current knowledge. During this trajectory, the expert is queried at each state encountered by the student policy, providing the correct action that the expert would have taken in this situation. These state-action pairs are then aggregated into a dataset used to retrain the student policy. This iterative process is repeated, with each iteration aiming to reduce the student policy's divergence from the expert policy. DAgger leverages the availability of the expert in simulation to achieve no-regret learning.

One can intuitively picture this with a simple example : In the naive implementation, the student must learn how to drive by watching videos of the expert driving, while with DAgger, the student is driving with the expert sitting next to him, correcting his mistakes.

The first iterations of the student policy are typically of poor quality, and so are the resulting trajectories. The student will benefit from expert demonstrations at the beginning of the training to improve convergence. To address this, DAgger introduces a policy selector function,$\beta(i)$, which determines whether the expert's or the student's action is applied to the environment. The policy selector function must respect two conditions : $\beta(0) = 1$ and $\beta(i) \to 0$ as $i \to \infty$. This means that initially, the expert's actions are used, but as training progresses, $\beta(i)$ gradually decreases towards 0, allowing the student's actions to be applied more frequently. Two functions are presented in the original paper that have also been assessed in this project:

$$\begin{aligned} \text{Indicator function :} \quad & \beta(i) = I(i) = \begin{cases} 1, & i = 0 \\ 0, & i > 0 \end{cases} \\ \text{Exponential Decay function :} \quad & \beta(i) = \alpha^i, \quad 0 < \alpha < 1 \end{aligned} \tag{5.2.1}$$

Differences from Standard DAgger A few adaptations are required due to the specific nature of the modified action space in this project. The expert actions can directly be applied to the environment, while the student actions must be decoded (with the action at time step zero selected) before they can be applied. In contrast, the expert actions must be encoded before being aggregated to the dataset. The flow of operation is illustrated in the figure 15.

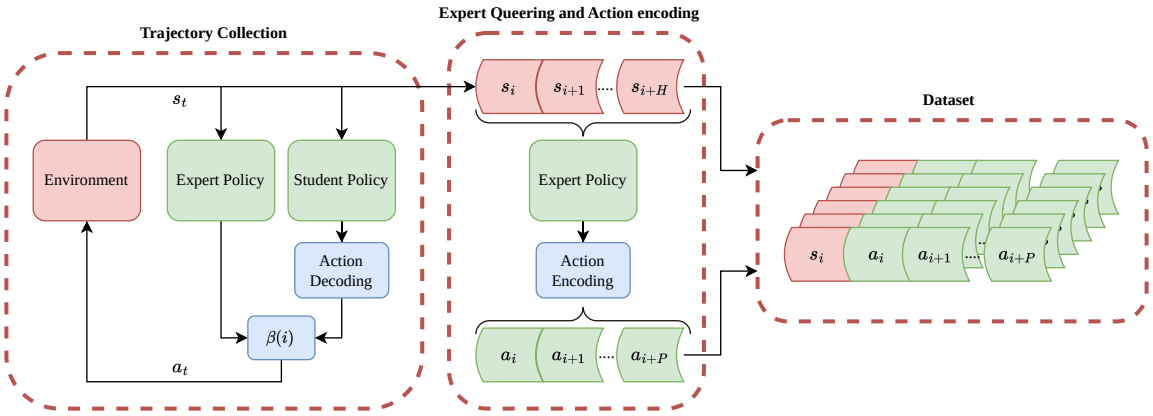


Figure 15: DAgger dataset generation procedure with modified action space

Algorithm 4 DAgger Algorithm with Modified Action Space

Require: Expert policy π_{exp}, policy selector function β, action encoding P and decoding P^{-1}

Initialize $\mathcal{D} \leftarrow \emptyset$
Initialize π_0
for $i = 1$ **to** N **do** ▷ Train for N Epoch
  Reset Action Buffer : $\mathcal{B}_A \leftarrow \emptyset$
  for $t = 1$ **to** T **do** ▷ Collect T trajectories
    Collect expert Action : $a_{t,exp} = \pi_{exp}(s_t)$
    Collect encoded student Action : $a_{stud} = \pi_i(s_t)$
    Decode student first action : $a_{t,stud} = P^{-1}(a_{stud})$
    Aggregate expert and student actions : $a_{t,agg} = \beta(i) \cdot a_{t,exp} + (1 - \beta(i)) \cdot a_{t,stud}$
    Append expert action to action buffer : $\mathcal{B}_A \leftarrow a_{t,exp}$
    Sample next observation : $s_{t+1} \leftarrow Env(s_t, a_{t,agg})$
  end for
  Encode expert actions : $a_{exp} \leftarrow P(\mathcal{B}_A)$
  Aggregate dataset : $\mathcal{D} \leftarrow \mathcal{D} \cup (s_0, a_{exp})$
  Train policy π_{i+1} on dataset $\mathcal{D}$: $\pi_{i+1} \leftarrow \text{Train}(\pi_i, \mathcal{D})$
end for
return π_N

In summary, DAgger provides a powerful framework for improving the performance of imitation learning by continuously refining the student policy based on the expert's feedback. By adapting the standard DAgger algorithm to accommodate our specific action space requirements, we ensure that the student policy can generate accurate and compatible actions with the sampling controller. This approach effectively reduces the regret associated with the naive implementation and allows for more robust policy learning in complex environments. The results of the different actions encoding, simulation, and network parameters on the student policy are presented and discussed in chapter 7.2.

5.3 Integrating the Trajectory Cost of the Sampling MPC into the Training of the Reinforcement Learning Policy

The other approach to obtain a policy with a compatible action space for the sampling controller is to train it directly in the right action space. To do so, the policy needs to be provided with the necessary feedback on each of its actions to learn an optimal behavior. The sampling model predictive controller is capable of doing this. It can evaluate the quality of actions by simulating a trajectory with its f model and associating a cost with its J cost function. By incorporating the trajectory cost into the RL policy's reward function, it is possible to provide feedback on all its actions. Enabling it to learn optimal actions without ever applying them to the system. In addition, the optimality of the problems re-assembles way more in this way, benefitting undoubtedly the transfer of the solution from one to the other.

This approach has been implemented and tested successfully on simple encodings. However, it would

have required tuning, which the time available for the project did not allow. Therefore, it will not be evaluated in more detail but is nevertheless presented as it is considered very promising.

6 Experimental Setup & Tools

This chapter presents the different tools used in this project, along with their advantages, drawbacks, and limitations. Then, it presents all the parameters used to obtain the results.

6.1 Tools

The implementation of the initial control scheme, before the start of the project, shown in figure 1 is written in Python and available on the public GitHub directory *iit-DLSLab/Quadruped-PyMPC*. This implementation includes various MPC controllers and is based on the *MuJoCo* simulator.

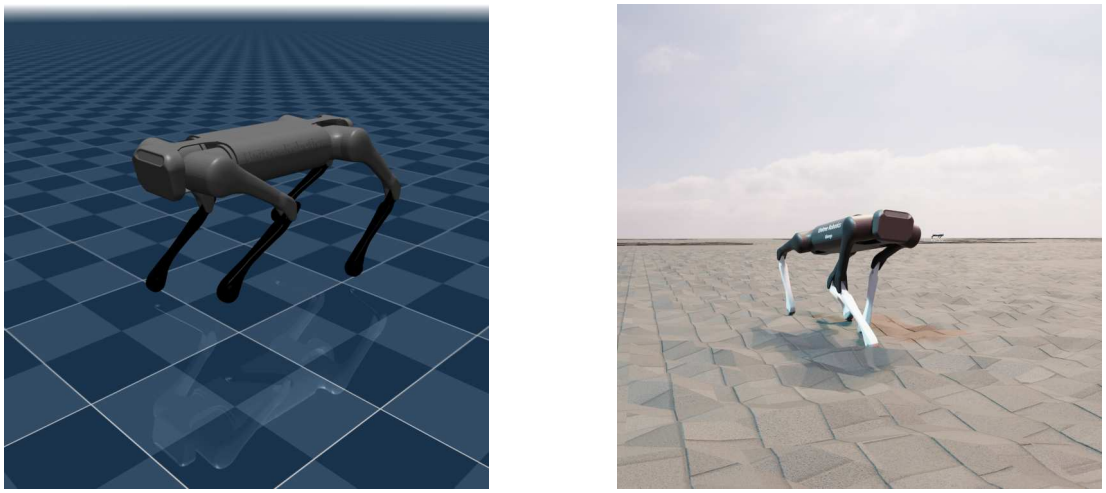
However, these elements had advantages and limitations, which, in some cases, needed to be adapted. The following chapters will discuss and justify the various choices.

6.1.1 Simulator

MuJoCo is a general purpose physics engine supported by *Google Deepmind* [Todorov et al. 2012]. It is fairly accessible, fast, and lightweight. Asserting the precision of a simulator is out of the scope of this project.

Nevertheless, *MuJoCo* was restricted to CPU use only at the time the project was launched [10], thus limiting its applications to machine learning that required GPU simulations to train a model in a reasonable time. As a result, we had to consider other simulators for training a Reinforcement Learning policy.

The natural choice was the simulator *IsaacSim* from *Nvidia*. It can be run entirely on the GPU and offers a wide range of machine learning tools. Its handling is considerably more complex and difficult to integrate into an existing framework. For this purpose, we used the *IsaacLab* framework developed by *ETH* and *Nvidia* [Mittal et al. 2023].



(a) *MuJoCo* (b) *IsaacSim*

Figure 16: Example of the robot *AlienGO* on the different simulators

These choices necessitated leaving the *iit-DLSLab/Quadruped-PyMPC* project to create a new one : *iit-DLSLab/dls_ isaaclab*. It was, therefore, necessary to re-implement the entire basic control algorithm (Figure 1), which was a major overhead. Nevertheless, these choices greatly facilitated the Reinforcement Learning part of the project, as we were able to rely on a strong, established, and documented structure with numerous pre-implemented functions.

[10] A first version of *MuJoCo* compatible with GPU was available but had no proper documentation and did not yet have all the features available. It was therefore left aside

6.1.2 Library

The computing library defines how all the problem variables will be processed on the machine, for which two choices have been considered: JAX [Bradbury et al. 2018] and PyTorch [Paszke et al. 2019].

The *Quadruped-PyMPC* implementation uses *JAX* for the Single Rigib Body Model (SRBM) and the sampling controller, achieving sub-millisecond solutions, which is incredibly fast. In contrast, the entire *IsaacSim* simulator is built using *PyTorch*. It is, therefore, impossible to avoid using *PyTorch*. We considered converting the tensors from one to the other, but there were a few difficulties with installation and debugging tools. In addition, despite its performance, *JAX* is very difficult to grasp, and few resources exist. The decision was therefore taken to use only *PyTorch* and to make the necessary re-implementations.

To train a Reinforcement Learning policy, *RSL_RL* was chosen [Hoeller et al. 2023a]. It is available by default with IsaacLab and features a PPO implementation entirely on the GPU.

6.2 Experimental Setup

Summary of the parameters of the experimental setup.

Parameter	**Value**
Learning rate	0.001
Learning rate decay γ	0.99
Number of iterations	10000
Number of learning epoch	5
GAE λ	0.95
clip ratio	0.2
Entropy Coefficient	0.005
Mini batch size	4
Desired KL	0.1
Number of env. step per iteration	24
Network initial noise σ	1.0

Table 6: The hyperparameters used for PPO in this project

Parameter	**Value**
Epochs	30
Policy Selector function	Exponential decay
Decay rate α	0.6
Collected trajectories per epoch	80000
Dataset maximal size	800000
Optimizer	Adadelta
Loss	Mean Squared Error
Batch size	64
Learning rate	1.0
Discount factor γ	0.7

Table 7: The hyperparameters for the DAgger algorithm in this project

Parameters	**Value**
Physics engine frequency	$200[Hz]$
Low-level control	$200[Hz]$
High-level control (RL)	$50[Hz]$
High-level control (Sampling MPC)	$100[Hz]$

Table 8: Hyper parameters for the simulator

index i	**State** x_i	**Weight** $Q_{i,i}$
1	x	0
2	y	0
3	z	111500
4	$\dot{x}$	5000
5	$\dot{y}$	5000
6	$\dot{z}$	200
7	ϕ	11200
8	θ	11200
9	ψ	0
10	$\dot{\phi}$	20
11	$\dot{\theta}$	20
12	$\dot{\psi}$	200
13-24	$p_{leg,xyz}$	0

Table 9: Diagonal weight of the Q matrix of the sampling MPC cost function

index i	**input** u_i	**Weight** $R_{i,i}$
1	$p_{FL,x}$	1000
2	$p_{FL,y}$	1000
3	$p_{FL,z}$	0
4	$p_{FR,x}$	1000
5	$p_{FR,y}$	1000
6	$p_{FR,z}$	0
7	$p_{RL,x}$	1000
8	$p_{RL,y}$	1000
9	$p_{RL,z}$	0
10	$p_{RR,x}$	1000
11	$p_{RR,y}$	1000
12	$p_{RR,z}$	0

index i	**input** u_i	**Weight** $R_{i,i}$
13	$F_{FL,x}$	0
14	$F_{FL,y}$	0
15	$F_{FL,z}$	0
16	$F_{FR,x}$	0
17	$F_{FR,y}$	0
18	$F_{FR,z}$	0
19	$F_{RL,x}$	0
20	$F_{RL,y}$	0
21	$F_{RL,z}$	0
22	$F_{RR,x}$	0
23	$F_{RR,y}$	0
24	$F_{RR,z}$	0

Table 10: Diagonal weight of the R matrix of the sampling MPC cost function

7 Results

The results will be analyzed in three parts, following the chronological order of development. First, the quality of the four policies trained by reinforcement learning will be assessed (chapter 7.1). Next, the transfer of these policies to a new action space through imitation learning and the influence of different parameters will be evaluated (chapter 7.2). Finally, the performance of these policies, when combined with the sampling controller, will be examined, addressing the two research questions (chapter 7.3 and 7.4):

- **RQ1** Is RL Warm Starting Beneficial to the sampling controller?
- **RQ2** Can Multiple Warm Starting Be Beneficial ?

7.1 Performance of the Reinforcement Learning policies

In this chapter, the performance of the four reinforcement learning policies trained to tackle the four defined tasks[11] will be evaluated. As a reminder, the tasks are :

- **Base Task:** omnidirectional speed control over simple terrain, without disturbances
- **Rough Task:** omnidirectional speed control over rough terrain, with high disturbances
- **Speed Task:** high forward speed command on simple terrain with low disturbance
- **Climb Task:** low forward speed command, on staircase-like terrain with low disturbances

The results presented in this chapter are the averages and variances over 50,000 trajectories.

7.1.1 Training performances

The learning algorithm's performance will not be discussed in detail as it is outside the scope of this project. Nevertheless, the policies have all converged between 1000 and 3000 iterations of the algorithm after tuning the reward function. It corresponds to a few hours of computation on the machine used in this project. The training gave satisfactory results, confirming the effectiveness of the algorithm.

7.1.2 Reward fitting

The first assessment is to determine how well the behavior generated by the policy aligns with the reward function. For this purpose, their performances were compared on a simple task, similar to one conducted under laboratory conditions : a low speed command ($v_x \in [0.3, 0.6] m/s, v_y \in [-0.2, 0.2] m/s, \omega_z \in [-0.5, 0.5] rad/s$) on flat ground with no disturbances. The reward function was common to all and identical to the one used to train the base policy. This simple task was chosen because it is within the distribution of the four different tasks, providing a consistent baseline for comparison.

As expected, the base task performed best, with the general-purpose tasks outperforming the specialized ones. Notably, the base task achieved at least 20% better performance than the other tasks, which is a considerable difference, given that this test is within the distribution for all the policies. The author hypothesizes that this discrepancy, referred to as the *out-of-tune* hypothesis, arises because the initial tuning was done at the first stage of the project, and subsequent changes to the implementation may have impacted this tuning. These changes include for example, modifications of the normalization scale, gravity compensation, and friction cone constraint imposition. The policies were re-tuned superficially, and the base policy may have benefited from more attention, explaining its relatively better performance.

7.1.3 Disturbance Reliability

The second evaluation focused on measuring the robustness of the different policies. To this end, two tests were conducted: a survival test with the same low-speed command as previously, but on very uneven terrain and with strong external disturbances, and a stair-climbing test, which in fact is identical to the climb task.

[11] Please note that the words base, rough, speed, and climb refer to the tasks, but also to the policies that have been trained on these tasks.

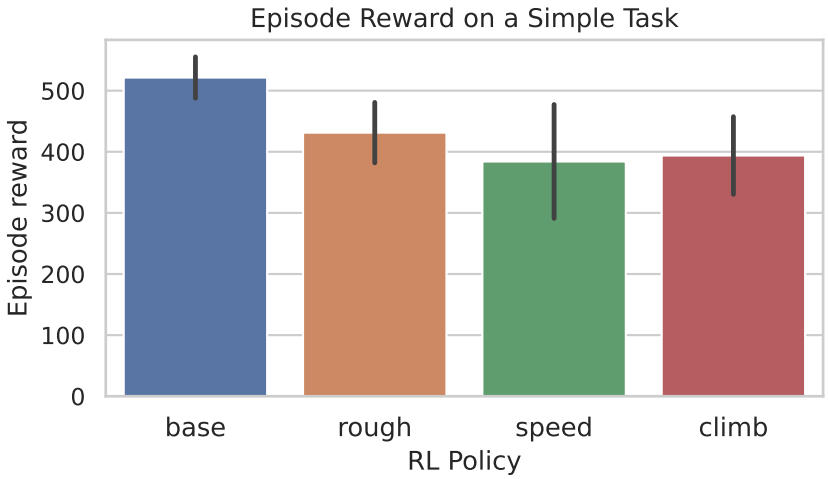


Figure 17: Mean episodic reward on a *simple* task between the four RL policies

The probability of survival is defined as the number of trajectories that completed the episode ($15s$) out of the total number of trajectories. An episode is terminated prematurely if there is contact with the robot's base or if it tilts more than 45 degrees. It is important to note that these two tests are within the rough policy distribution, and the staircase test falls within the climb policy distribution. Otherwise, they are tests outside the distribution encountered during training.

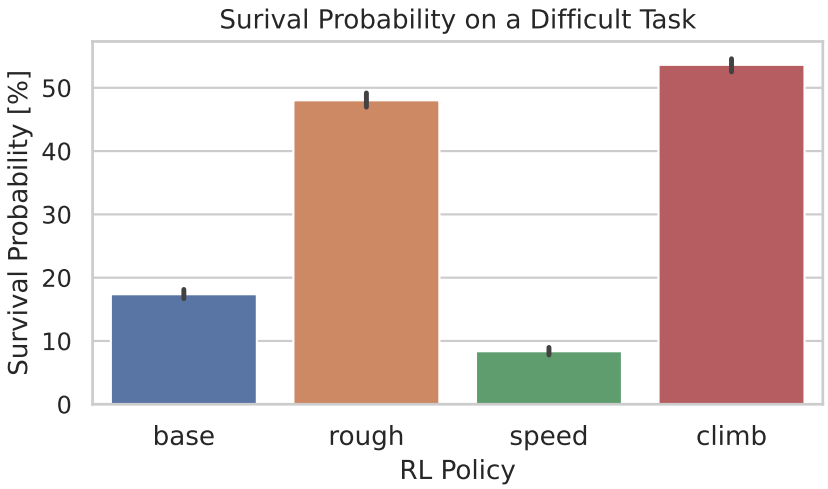


Figure 18: Survival rate on a *survival* task for the different RL policies

Unsurprisingly, neither the base nor the speed policy was able to withstand perturbations in unfamiliar terrain. Nevertheless, the excellent performance of the climb task is noteworthy, as it generalized its recovery movements, train on staircases, to unfamiliar terrain (box, rugged, and sloping terrain as seen in Figure 12) and to higher, previously never experienced disturbances (randomization of the friction coefficient and pushes). It even outperforms the rough task.

The results for the staircase task exposed in figure 19 were not particularly good. Despite the rough task, having seen stair-like environments during training, it did not perform well. The performances dropped off after only $25cm$, even for the climb task. The suggested hypothesis is again the *out-of-tune* hypothesis for these two tasks, as they had performed better.

Furthermore, the problem formulation is not well suited to stair climbing. As the steps are particularly tight, the robot should be able to turn to pass one leg and then the other, as the legs are often kinematically blocked against the stairs. However, this behavior is penalized because it has to track a yaw speed ω_z

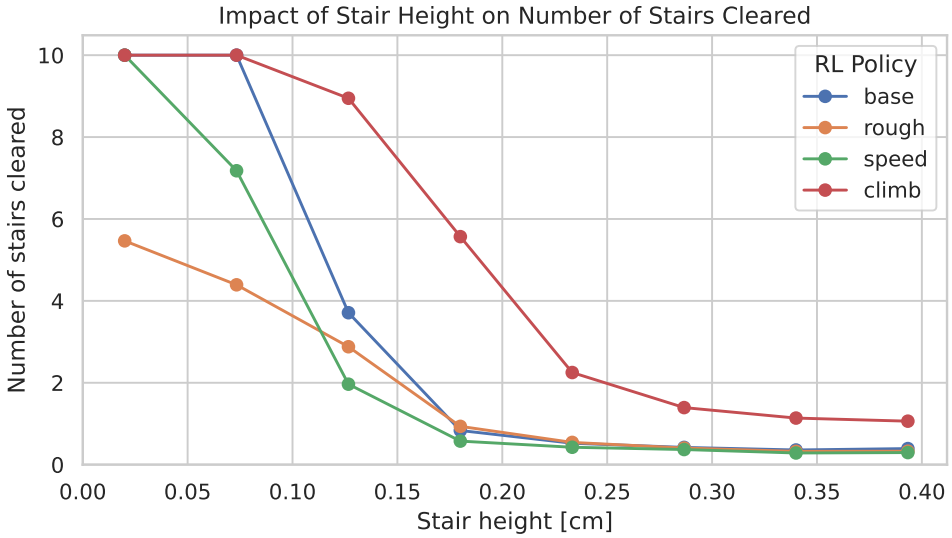


Figure 19: Performance comparison in staircase across the different policies

reference. The relaxation in the speed command v_x introduced by the soft square exponential kernel (appendix D) increased the performances by about $5cm$. Despite this flexibility, a speed command seems not very suitable for this problem and way point navigation would have been more appropriate.

7.1.4 Locomotion, Energy Efficiency, and Gait

Finally, the third assessment of interest is the locomotion capabilities of the policies, performed on flat terrain without disturbances.

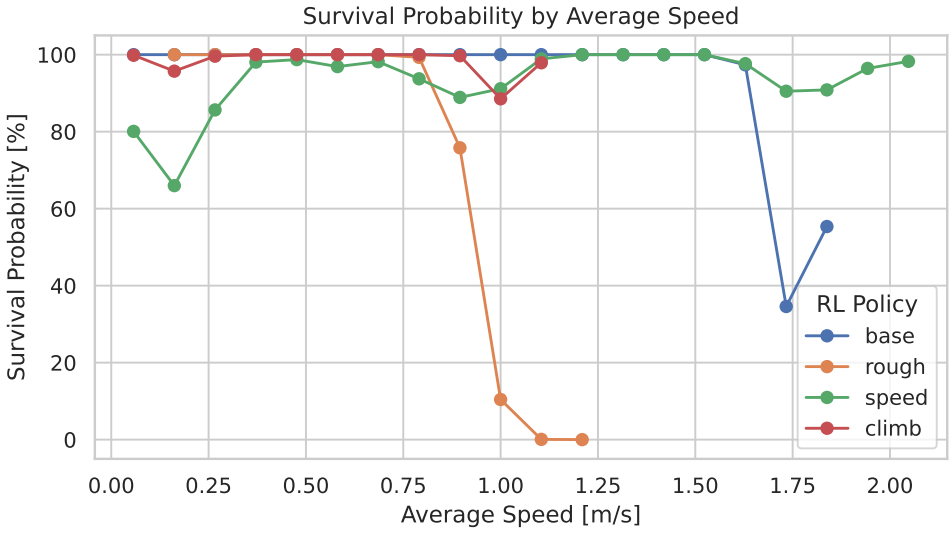


Figure 20: Survival Probability comparison between the RL policies at different (forward) speed

The survival probability as a function of speed is shown in figure 20 and there are several interesting things to note. The speed policy is the one that can go the fastest, but it is also the least reliable in its distribution. Next, there is a remarkable generalization of the base policy, which can reach almost twice the speed at which it was trained. In addition, the difference between the rough policy and the climb policy when leaving their distributions should be noted. The rough policy falls immediately as soon as the speed exceeds the $0.8m/s$ seen during training. In contrast, the climb policy, although unable to reach high speeds, does not fall, preferring to violate the speed command rather than fall.

Figure 21 illustrates the ability of the different policies to follow an omnidirectional speed command.

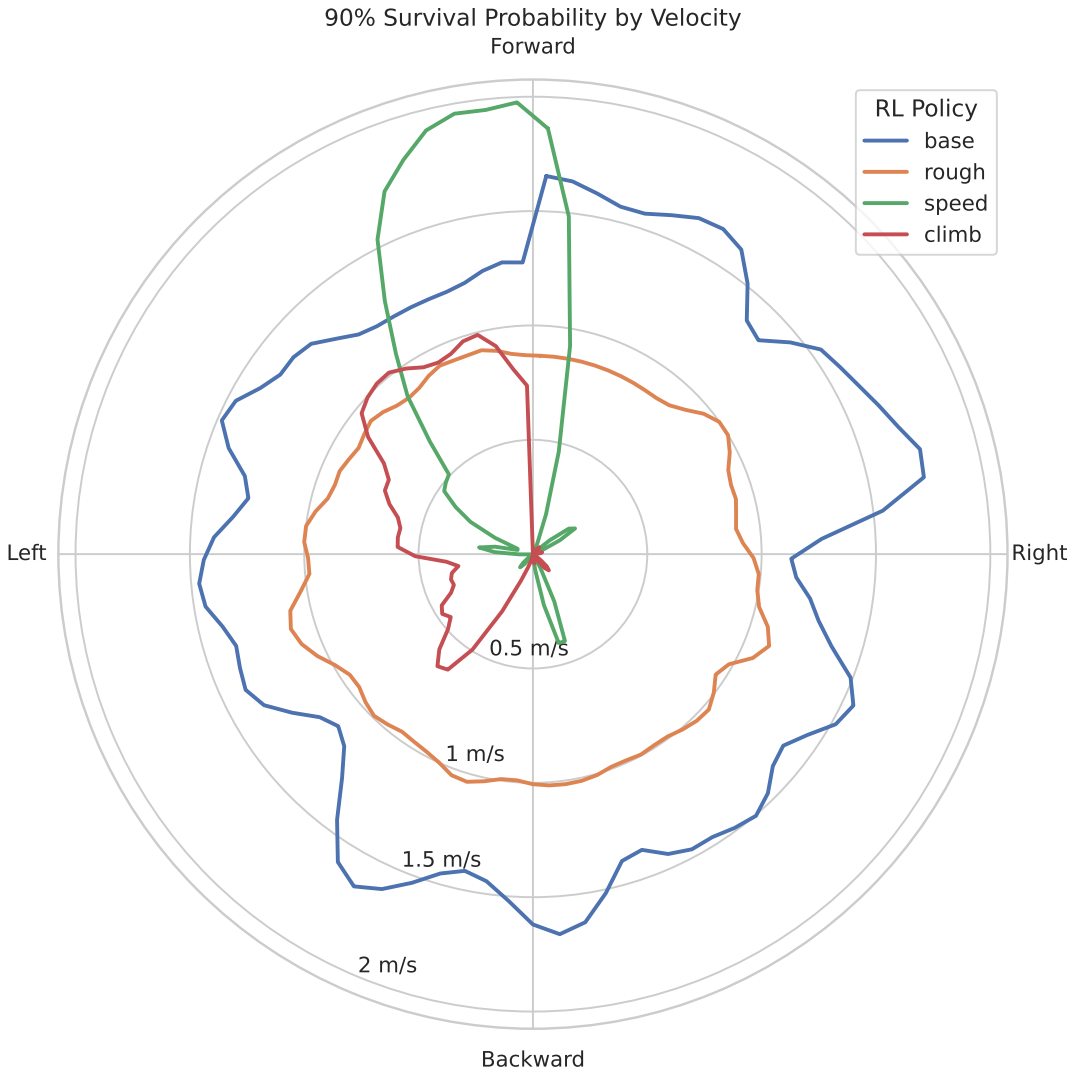


Figure 21: Omnidirectional capabilities comparison between the different tasks as isoline where 90% of the samples succeed the task for a given velocity and direction

We can see that the rough, speed, and climb policy corresponds relatively well with the speed distributions they saw during training. However, the base policy is able to go 50% faster in all directions, whereas it has never seen a speed command greater than $1m/s$ during training.

The Cost of Transport (CoT) of the different policies is evaluated in figure 22. The results observed are similar to those found in *Fast and Efficient Locomotion via Learned Gait Transitions* [Yang et al. 2021] for the *Unitree A1* robot, with a CoT ranging from 2 at $0.5m/s$, down to slightly less than 1, at speed $2.5m/s$.

Moreover, a very interesting behavior can be observed in the speed policy. It has the greatest penalty on the CoT and should, therefore, minimise it as much as possible. Nevertheless, up to $1.1m/s$ its CoT is higher than for the base policy. An explanation could be the disturbances during training and a distribution biased towards higher velocity. On the other hand, from $1.1m/s$ onwards, the CoT of the speed policy drops to a particularly low level. This corresponds to the moment when it changes gait from a trot to a hybrid trot-gallop.

Figures 23a and 23b compare the gaits of the base policy and the speed policy at different speeds. On the one hand, the base policy learned a very regular trot, which enabled it to generalize its gait to speeds it had never experienced before. On the other hand, the speed policy has also learned a trot at low speeds, but from 1.1m/s, it transitions to a trot-gallop hybrid. The front legs (FL, FR) trot, while the rear legs (RL, RR) gallop. The frequency also increases drastically at its maximum. Although interesting, this

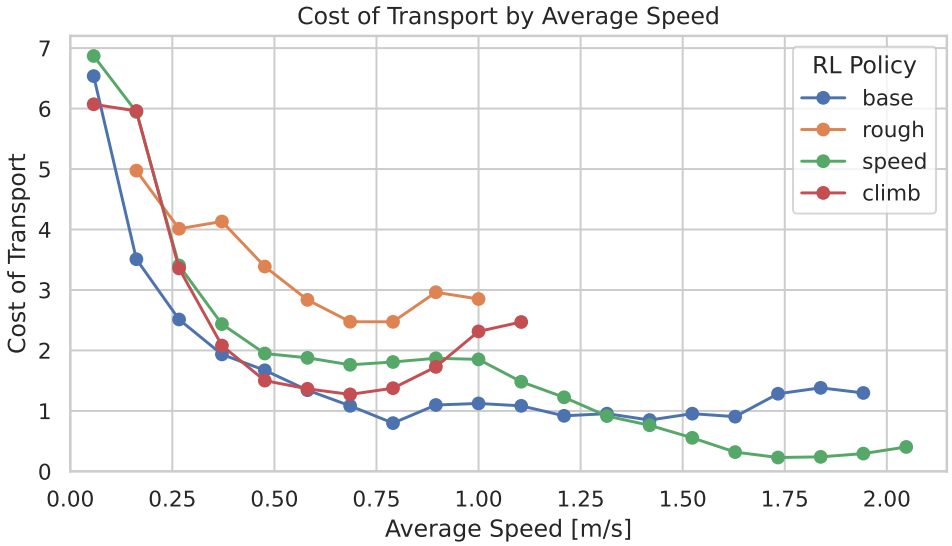


Figure 22: Comparison of the Cost of Transport between different policies

gait would certainly not be transferable to practical use, due to the excessively high stepping frequency. Furthermore, the speed policy trot is less efficient than the base policy trot. This can be explained by the fact that speed distributions are biased towards high speeds and the training is certainly *out-of-tune*.

Another thing to note is that no policy has learned a flying gait, even at very high speed. There are several reasons for this. On the one hand, the height tracking reward is based on the proprioceptive height which is not defined when no legs are in contact, in which case it is the average distance of the legs from the base that is taken into account. This distorts the proprioceptive height and is therefore penalized. In addition, a maximum force and a relatively restrictive friction cone are imposed, thus limiting acceleration capacity. Therefore, in order to have sufficient acceleration, the pushing time must be maximized by keeping the legs on the ground.

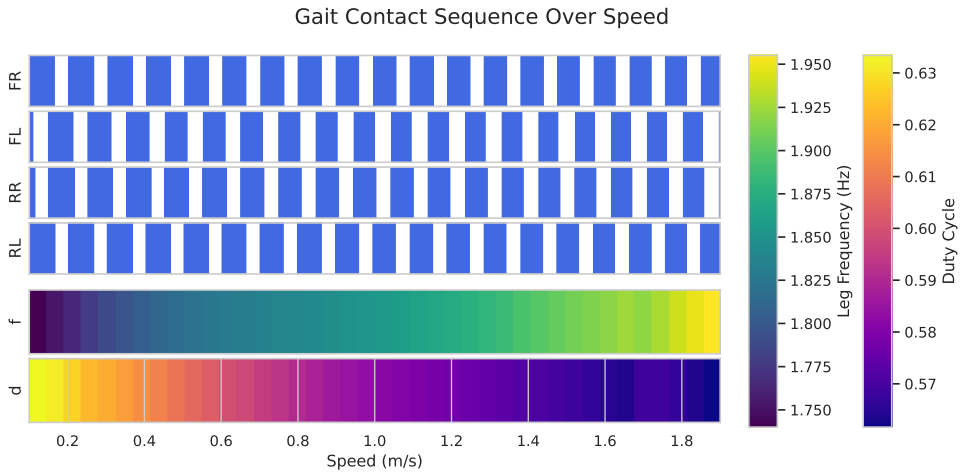
7.1.5 Conclusion

A few conclusions can be drawn from these results. Firstly, the results obtained are within the bounds of what can be expected from *AlienGo*, which confirms the approach adopted. Secondly, one important conclusion that can be drawn is that the quality of the behaviors generated is more important for generalization than the diversity of the situations in which they are confronted.

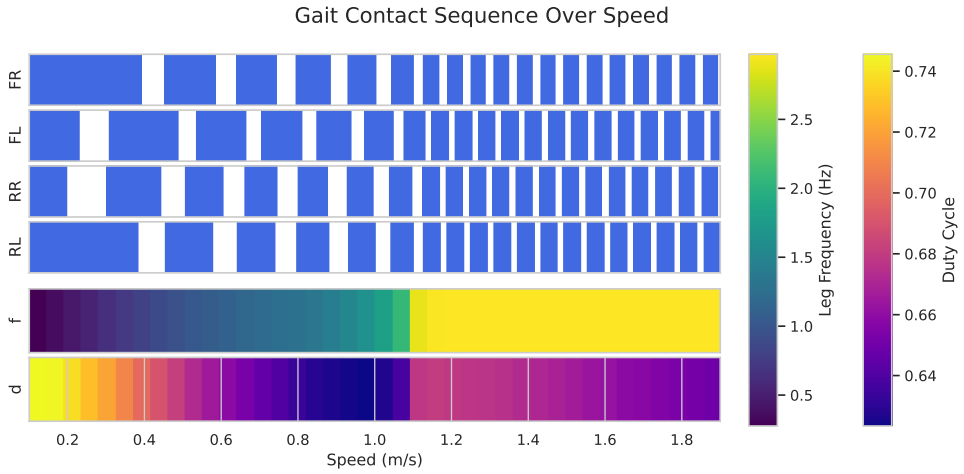
This was observed with the trot of the base policy, which was extremely stable and enabled it to generalize at speeds that it had not trained for. In the same way, the recovery movements of the climb policy, trained on difficult obstacles but with limited diversity and few disturbances, are quality movements. It was able to generalize these behaviors to disturbances and terrain never experienced before, surpassing the rough policy in terms of robustness.

This corroborates with the conclusions of *learning quadrupedal locomotion over challenging terrain* by [Lee et al. 2020]. They state that *that simulating the astonishing variety of the physical world may not be necessary* and that a controller trained by RL can successfully generalize to unseen conditions if trained right. Furthermore, it suggests that the disturbances and the curriculum learning parameters of the rough policy were too aggressive, preventing it from correctly learning quality movement first.

Finally, the ambitious approach of this project was to some extent prejudicial to these results, as some of the policies would certainly have performed better if more consideration had been given to their tuning.



(a) Base policy



(b) Speed policy

Figure 23: Comparison of gaits across velocity for the base and speed policy

7.2 Distillation of an Export Policy into a Student Policy

In chapter 5.2, Imitation Learning was presented, and two algorithms were introduced: a naive implementation and the dataset aggregation technique, DAgger. First, a brief comparison of the performance of these two algorithms will be done. Then, an in-depth examination of DAgger will be conducted by systematically exploring the impact of mini-batch size, policy selector function, and especially the different action encoding. A discussion on specific problems and how they have been assessed will also be done.

7.2.1 Metrics and Evaluation Method

All the results presented in this chapter are based on the same expert policy, a policy trained on the *base* task with the method introduced in chapter 4 giving excellent results in simulation. Its spaces are single observation, single action. Every Mean Squared Error (MSE) results presented in this chapter are based on a testing set not seen during training. The testing set has been collected on the *base* task and consists of 50'000 trajectories with the horizon and actions encoding required by the student.

$$MSE(y, \hat{y}) = \frac{\sum_{i=1}^{N}(y - \hat{y})^2}{N} \tag{7.2.1}$$

The mean squared error allows us to assess the quality of the data fitting between the prediction of the student $\pi(x) = \hat{y}$ and the the test set y. It provides a baseline for evaluating the quality of the network. However, beyond a certain point, MSE may not fully capture the policy's ability to solve the problem, in this case, navigating through an environment. The challenge lies not merely in data fitting, but in understanding and replicating the underlying recovery behaviors and mechanisms of the expert policy to effectively solve the task.

To address this, a new metric is introduced, namely *simulation performance* SP. The first challenge of this new metric is that it should provide a comparison as fair as possible between the different parameters (horizon, time step) and the action encoding type (spline parametrization or discrete action). For this reason, only the decoded first action is used, which, ultimately, all configurations are able to provide. With this action, the simulation is stepped and the step reward (of the *base* task) is collected. A sufficient amount of trajectories $N = 50'000$ with a sufficient horizon $T = 15s$ are collected to average out the influence of the random first state and other effects. These cumulative rewards r_{stud} are then re-scaled between the rewards achieved by the expert r_{exp} and the untrained policy r_0 under the same conditions. This gives a value between 0 and 1[12], which can be interpreted as a percentage of the reward achieved by the expert.

$$SP(r_{stud,r_exp,r_0}) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{T} r_{stud,i,j} - \sum_{i=1}^{N}\sum_{j=1}^{T} r_{0,i,j}}{\sum_{i=1}^{N}\sum_{j=1}^{T} r_{exp,i,j} - \sum_{i=1}^{N}\sum_{j=1}^{T} r_{0,i,j}} \tag{7.2.2}$$

7.2.2 Comparison of Imitation Learning Naive Implementation and DAgger

For completeness, a comparison between the Naive Implementation and DAgger algorithms was conducted to evaluate their performance based on key metrics: Mean Squared Error (MSE) and simulation performance, summarized in Table 11. For both algorithms, the parameters are 20 epochs, mini-batch size of 64, a time horizon $H = 5$, a discretization time $dt = 0.02[s]$, and the exponential decay function 0.6^i for DAgger.

Algorithm	Encoding (p, F)	MSE	SP
Naive	discrete, discrete	0.1546	68.49 %
DAgger		0.2185	98.91 %
Naive	first, spline	0.1613	55.62 %
DAgger		0.1870	99.30 %

Table 11: Performance between the imitation learning naive implementation and DAgger

The MSE are relatively small, meaning the different networks were able to fit the training data well. As expected, the MSE is smaller in the naive implementation because the training data closely

[12]Not rigorously true, the student could eventually be better than the expert or worse than an untrained policy

resemble the testing data, as both consist solely of expert actions. In the DAgger case, the training data also contains expert action obtained on student trajectories. Although the data fitting is better in the naive implementation, this does not translate to a better policy when tested in simulation. The simulation performance of the naive implementation is noticeably worse, which corroborates with the concept of *regret* introduced earlier (chapter 5.2.2). The policy is shaky, struggles to walk, and falls after a few seconds. On the other hand, one cannot distinguish visually the expert policy from the student in simulation when trained using DAgger and the student achieved up to 99% of the reward of the expert (simulation performance SP metric). The conclusion is unequivocal: the DAgger algorithm greatly improves the performance of the trained policy.

7.2.3 Detailed Analysis of DAgger Results

Influence of Mini Batch Size : The influence of the mini-batch size on DAgger algorithm has been assessed on the performance and convergence speed. The results are presented in table 12. As a reminder, the dataset size and the number of epochs remain constant. This means the bigger the mini-batch size, the fewer policy updates (gradient updates) will be done.

Performance tends to decrease as the batch size increases. This can be explained by the reduced number of policy updates per epoch for larger batch sizes. For a perfectly fair comparison, other hyper-parameters like the learning rate decay should ideally be adjusted as well. Nonetheless, this evaluation allows us to assess the level of *stochasticity* introduced by different batch sizes. Even with the smallest batch size, the policy update is accurate enough to ensure convergence while reducing the number of epochs required and, thus, the training time. The best results have been achieved with a batch size of 64, which will be the batch size used for the rest of this project.

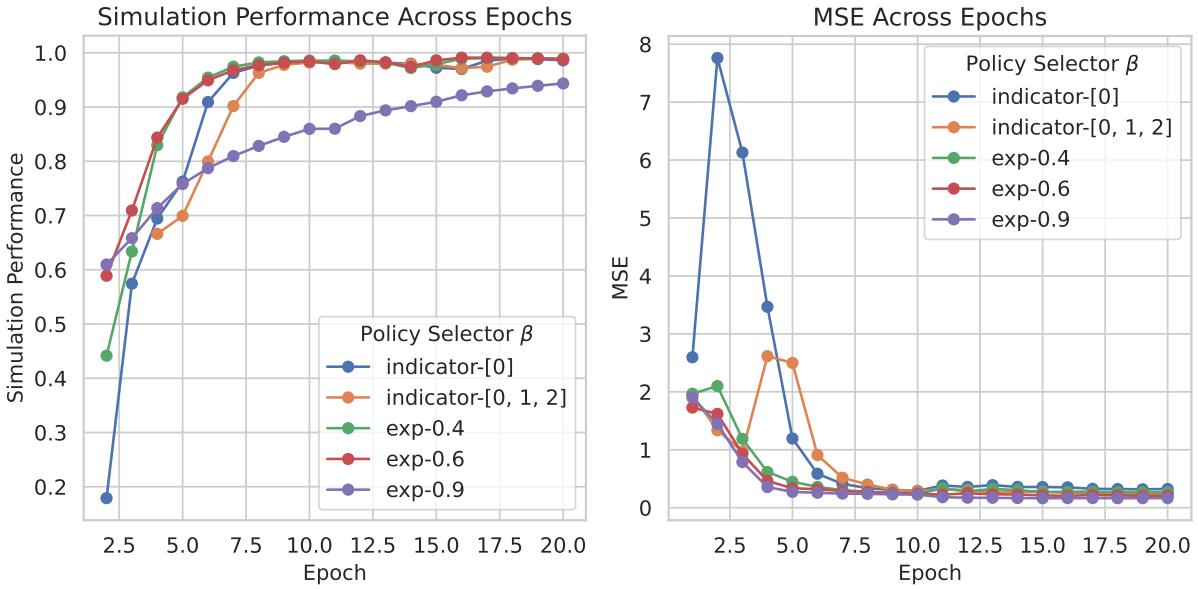
Mini Batch Size	64	128	256	512	1024
MSE	0.2048	0.2534	0.2764	0.3444	0.4876
SP	98.53%	98.54%	97.79%	94.89%	89.60%

Table 12: Batch Size influence on MSE and simulation performance - Horizon $H = 5$, Encoding (p =discrete, F =discrete)

Influence of the Policy Selector Function : The DAgger paper [Ross et al. 2010] introduced two types of policy selector functions: the indicator function and the exponential decay function, as presented in Equation 5.2.1. Their effect on the simulation performance and mean squared error on test-set are illustrated in figures 24a and 24b respectively.

The policy selector function that demonstrated the best performance is the exponential decay with the hyperparameter $\alpha = 0.6$, which is consistent with the result presented in the original paper. It is the right compromise between sufficient expert demonstration and student experimentation and will be the policy selector function used in the rest of this project.

Less expert demonstration (with indicator function or smaller α) shows difficulty in the early phase of training but eventually catches up. However, too much expert demonstration affects simulation performances negatively, even though the MSE is lower. This effect is similar to the effect observed in the comparison between the naive implementation and DAgger in chapter 7.2.2.



(a) Simulation Performance comparison between different policy selector functions

(b) Mean squared error comparison on test set between different policy selector functions

Figure 24: Performance assessment of the different policy selector functions for the DAgger algorithm

Influence of Action Encoding : The Influence of the different Action Encoding will be presented and discussed in this chapter. Let's begin with a recall of the different action encoding. The dataset consist of H expert actions (f_i, d_i, p_i, F_i) along an horizon, that will be encoded as $(f_0, d_0, (p'_0, ...), (F'_0, ...))$. The different encoding for p, the feet touch down positions, and F, the ground reaction forces, were presented in chapter 5.1. For simplicity, the following nomenclature will be used :

- **discrete** : The sequence is not encoded, and the complete sequence is kept: $\{x_i\}_{i\in H} \rightarrow \{x_i\}_{i\in H}$
- **spline** : The sequence is encoded with cubic spline parameters : $\{x_i\}_{i\in H} \rightarrow \{\theta_i\}_{i\in[0,3]}$
- **first** : The sequence is encoded by keeping only the first element: $\{x_i\}_{i\in H} \rightarrow x_0$

The performance of every different combination has been assessed and is summarized in Table 13. This provides a reference point for evaluating them. Based on this, a more in-depth analysis will be conducted afterward. Firstly, one can note that the MSE is fairly low, meaning the network is able to fit decently the data, giving evidence that the type of network and the learning algorithm are well suited for the problem.

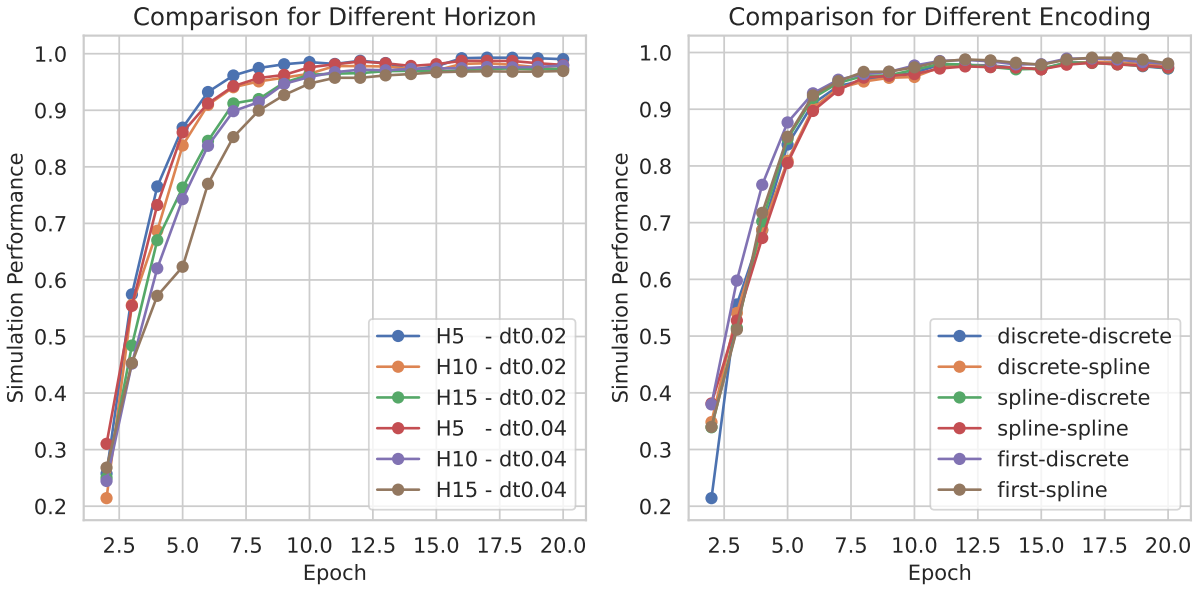
In addition, the MSE for *spline* is systematically slightly higher than for *discrete*, and the divergence increases with the horizon. This is expected as two of the four parameters of the spline capture the curvature of the spline, which have typically a higher variance in the dataset, making learning a spline a more difficult problem. However, the MSE remains reasonably low and the simulation performance high, meaning that the network was able to fit the data and provide meaningful actions.

However, a significant drop in performance is observed for foot touch-down position p with *spline* and a long prediction horizon. This can be explained because two touch-downs arise during the horizon, and the spline is not able to capture the trend anymore. One can conclude from this that *spline* encoding for p is probably not a meaningful choice.

On the other hand, *spline* encoding seems appropriate for the ground reaction forces F. The simulation performance decays only slightly with the prediction horizon, indicating the ability of this parametrization to capture the trend across time while significantly reducing the number of parameters.

N°	Action Encoding (p, F)		dt $[s]$	Horizon	Time H. $[s]$	Dim.	MSE	SP
1.	discrete	discrete	0.02	5	0.1	108	0.2193	98.53%
2.	discrete	spline	0.02	5	0.1	96	0.2181	98.61%
3.	spline	discrete	0.02	5	0.1	100	0.1817	99.11%
4.	spline	spline	0.02	5	0.1	88	0.1846	98.73%
5.	first	discrete	0.02	5	0.1	76	0.1428	98.98%
6.	first	spline	0.02	5	0.1	64	0.1548	99.06%
7.	discrete	discrete	0.02	10	0.2	208	0.2584	97.59%
8.	discrete	spline	0.02	10	0.2	136	0.3241	97.39%
9.	spline	discrete	0.02	10	0.2	160	0.2701	97.07%
10.	spline	spline	0.02	10	0.2	88	0.3792	97.43%
11.	first	discrete	0.02	10	0.2	136	0.1669	98.51%
12.	first	spline	0.02	10	0.2	64	0.2282	98.70%
13.	discrete	discrete	0.02	15	0.3	308	0.3186	95.97%
14.	discrete	spline	0.02	15	0.3	176	0.3948	96.30%
15.	spline	discrete	0.02	15	0.3	220	0.4067	95.31%
16.	spline	spline	0.02	15	0.3	88	0.6288	95.60%
17.	first	discrete	0.02	15	0.3	196	0.2049	97.91%
18.	first	spline	0.02	15	0.3	64	0.2798	97.87%
19.	discrete	discrete	0.04	5	0.2	108	0.2007	98.44%
20.	discrete	spline	0.04	5	0.2	96	0.3132	97.93%
21.	spline	discrete	0.04	5	0.2	100	0.2230	98.44%
22.	spline	spline	0.04	5	0.2	88	0.3311	97.96%
23.	first	discrete	0.04	5	0.2	76	0.1427	98.74%
24.	first	spline	0.04	5	0.2	64	0.2242	99.08%
25.	discrete	discrete	0.04	10	0.4	208	0.3531	96.96%
26.	discrete	spline	0.04	10	0.4	136	0.4070	97.34%
27.	spline	discrete	0.04	10	0.4	160	0.4574	95.34%
28.	spline	spline	0.04	10	0.4	88	0.5933	95.01%
29.	first	discrete	0.04	10	0.4	136	0.2013	98.19%
30.	first	spline	0.04	10	0.4	64	0.2666	98.19%
31.	discrete	discrete	0.04	15	0.6	308	0.3788	95.63%
32.	discrete	spline	0.04	15	0.6	176	0.5386	95.61%
33.	spline	discrete	0.04	15	0.6	220	0.4597	85.50%
34.	spline	spline	0.04	15	0.6	88	0.7303	86.75%
35.	first	discrete	0.04	15	0.6	204	0.2099	98.04%
36.	first	spline	0.04	15	0.6	64	0.2680	96.13%

Table 13: Imitation Learning Results



(a) Comparison between different Horizon H and dt for *discrete-discrete* encoding

(b) Comparison between different encoding for $H = 10$ and $dt = 0.02s$

Figure 25: Simulation performance comparison for imitation learning policies with different parameters

7.2.4 Problem of learning spline parameters

Learning the spline parameters was particularly challenging and required an adaptation of the implementation. Indeed, with the parameterization used, the first and last parameters represent the curvature, whereas the others are points on the curve. In the expert's data, the variance of the curvature is typically an order of magnitude greater than that of the points on the curve. This difference in scale considerably affects learning. To address this, these two parameters were rescaled (by a factor of 10), which solved (or at least mitigated) the problem and enabled the network to learn the spline parameters correctly.

7.2.5 Conclusion

In general, the DAgger algorithm provided good policies that were able to learn a new action space from an expert efficiently, no matter the type of encoding or the prevision horizon. The results were satisfactory and as good as one could have wished at this stage of the project. Rest to assess how these results will transpose to the sampling controller, which will be the subject of the next chapter.

7.3 Effect of the Warm Start from a Reinforcement Learning Policy on the Sampling Controller

This chapter evaluates the impact of the warm start parameters on the sampling controller. In particular, the actions' encoding, the horizon, the number of samples, and the optimization of the different variables (f, d, p, and F) will be evaluated. In this way, an optimal configuration can hopefully be found.

7.3.1 Metrics and Evaluation Method

All values presented in this chapter are median, and the error bars represent the first and third quartiles. When the simulation is reset, aberrant values are generated, completely distorting the trajectory's cost. The choice of the median is therefore justified, as it is robust to outliers and avoids this problem being reflected in the results, which the mean, for example, cannot do. In addition, a set of metrics will be introduced for this chapter.

The RL Reward will be used to measure the optimality of a configuration with respect to the reinforcement learning policy. It is defined as the instantaneous reward calculated by the reward function of the base task (equation 4.3.1). The higher this value is, the better the robot performs, according to RL policy criteria.

The Initial Cost measures the optimality of the robot's behavior with respect to the optimality of the sampling controller. It is defined as the cost for the current state of the robot, calculated with the sampling controller's cost function (equation 3.2.1). The lower this value, the better the robot performs according to the sampling controller's criteria.

The Rollout Cost reflects the optimality of the solution found by the sampling controller. It is defined as the cost of the optimal trajectory. The lower this value, the more optimal the solution is according to the sampling controller's criteria. The last metric used is the survival rate, as defined earlier.

The performance will be compared between three controllers. First, a controller warm started by an IL policy trained by an RL expert policy. For simplicity, this will be referred to as *RL warm start*. Then, a second controller warm started by the previous solution, simply called *naive warm start*. Finally, an RL policy alone, called *RL baseline*, in order to provide a reference for comparison. In the entire chapter 7.3, the expert for IL policy, as well as the RL baseline, are based on the RL base policy introduced earlier.

Moreover, collecting a trajectory with the sampling controller is time-consuming, as the implementation does not allow parallelization of the environment (rollouts are already parallelized). Therefore, the number of trajectories collected is reduced from 50,000 to 100 for experiments with the sampling controller, corresponding to around 40 minutes of simulation. Finally, trajectories are collected on the simple task defined in chapter 7.1.2 if not otherwise specified.

Finally, it is important to note that, unless explicitly stated otherwise, the sampling controller only optimizes the ground reaction forces. Gait and footholds are taken directly from the RL policy for the *RL warm starting*. For the naive warm starting, they come from a handcrafted trot and a heuristic foothold generator. The optimization of the latter is complex and will be discussed specifically.

7.3.2 Effect of the Action Encoding on the Performances

The first comparison is to find the most effective action parameterization, illustrated in table 14 and figure 26. It confirms the author's intuition that the most suitable parameterization is *first-spline*, i.e., a unique foot touch-down position p and four spline parameters for ground reaction forces. The results suggest that restricting the search space is more important than having finer control over the actions.

In addition, several interesting effects are worth noting. On the one hand, the proposed warm start significantly reduces variance, which is more important for robustness than lowering the solution's cost. On the other hand, it also makes it possible to consider parameterizations that would not work without this approach. For example, the cost of the *discrete-discrete* parametrization has been divided by 15

	Action Encoding (p, F)		RL Reward	Initial Cost	Rollout cost
RL warm starting	discrete	discrete	2.320	61.44	787.3
	discrete	spline	2.336	46.28	485.0
	spline	discrete	2.325	47.56	581.9
	spline	spline	2.333	45.72	455.6
	first	discrete	2.339	35.15	432.8
	first	spline	**2.347**	**30.72**	291.8
Naive warm starting	discrete	discrete	1.297	879.7	2919
	discrete	spline	2.259	33.26	221.5
	spline	discrete	1.331	854.9	8107
	spline	spline	2.266	32.46	218.4
	first	discrete	1.242	983.7	9576
	first	spline	2.259	31.43	**211.7**
RL baseline			**2.365**	86.45	

Table 14: Comparison of sampling controller performance across different action encoding with horizon $H = 10$, and MPC discretization time $dt = 0.02[s]$. All values are median value

between the two warm starts and offers decent locomotion.

Furthermore, there are several optimality differences. For example, the initial cost and rollout cost do not match up. The RL baseline obtains the best RL reward and is only tenth in the initial cost ranking. This difference in optimality can be explained by the different functions (quadratic or exponential) used to calculate the cost and by the different weights.

Interestingly, the rollout cost does not give the same ranking as the initial cost either. For *first-spline* encoding, the naive warm start solution has a lower cost than its counterpart, meaning its predicted actions are more optimal. Nevertheless, its performances are not as good. The proposed explanation is that the naive warm start, less guided in its solution, manages to lower the cost more along the trajectory. In contrast, the RL warm start controller is guided by a more optimal solution at the start of the trajectory but deteriorates as the trajectory progresses due to differences between the simulator and the model.

The difference in trend between the rollout cost and the initial cost highlights the difference in cost evolution along the prediction horizon and between the sampling controller's f model and the simulation. In the rest of this project, the action encoding *first-spline* will be used.

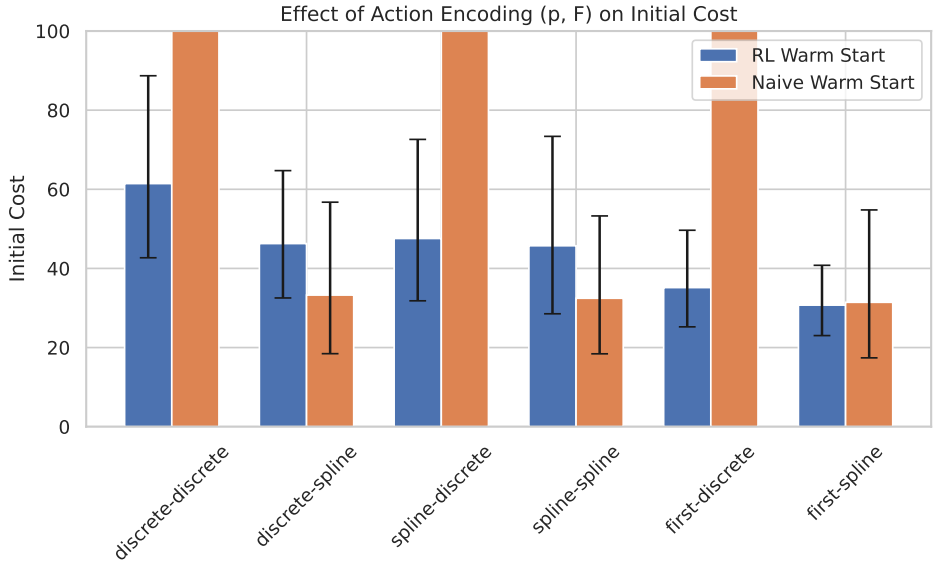


Figure 26: State cost comparison across the different encoding with horizon $H = 10$, and MPC discretization time $dt = 0.02[s]$. All values are median value

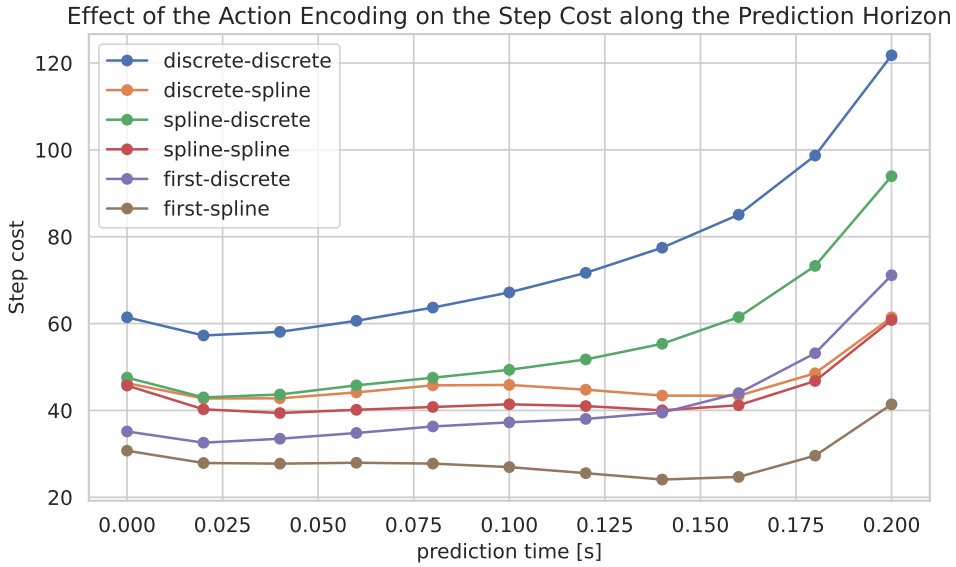


Figure 27: Rollout cost comparison across the different encoding with horizon $H = 10$, and MPC discretization time $dt = 0.02[s]$, with RL warm starting. All values are median value

7.3.3 Effect of the Prevision Horizon on the performances

	dt $[s]$	Horizon	Time H. $[s]$	RL Reward	Initial Cost
RL warm starting	0.02	5	0.1	**2.355**	**14.62**
	0.04	5	0.2	2.348	25.04
	0.02	10	0.2	2.346	32.93
	0.02	15	0.3	2.329	75.92
	0.04	10	0.4	2.329	128.4
	0.04	15	0.6	2.238	477.2
Naive warm starting	0.02	5	0.1	2.258	31.41
	0.04	5	0.2	2.255	31.41
	0.02	10	0.2	2.248	28.61
	0.02	15	0.3	2.233	41.11
	0.04	10	0.4	2.179	79.81
	0.04	15	0.6	1.756	408.8
RL baseline				**2.365**	86.45

Table 15: Effect of the prediction horizon on the sampling controller performance, with (*first-spline*) encoding. All values are median values.

The impact of the prediction horizon is presented in table 15. The performance of all metrics deteriorates with the horizon increasing. In addition, figure 28 illustrates the cost explosion of the latest predictions on a logarithmic scale. Regardless of the horizon, the latest costs systematically explode. The author cannot explain this behavior. Intuitively, one would expect the opposite effect: the MPC should optimize the trajectory to decrease costs along the horizon. The cost explosion could be due to an implementation error, but this behavior has been observed identically on a separate implementation made by Turrisi et al. on *MuJoCo* [Turrisi et al. 2024]. The few degrees of freedom of the optimization (a single spline and a single foot touch-down position) could also explain the phenomenon. However, the same behavior was observed with all encodings on both implementations. This is a limitation of the sampling controller and should be further investigated in detail.

In the remainder of this project, a horizon $H = 10$ and a discretization time $dt = 0.02s$ will be used. The effect of cost explosion is contained, while a slightly longer time horizon allows the controller to perceive the effect of gait and foothold optimization more.

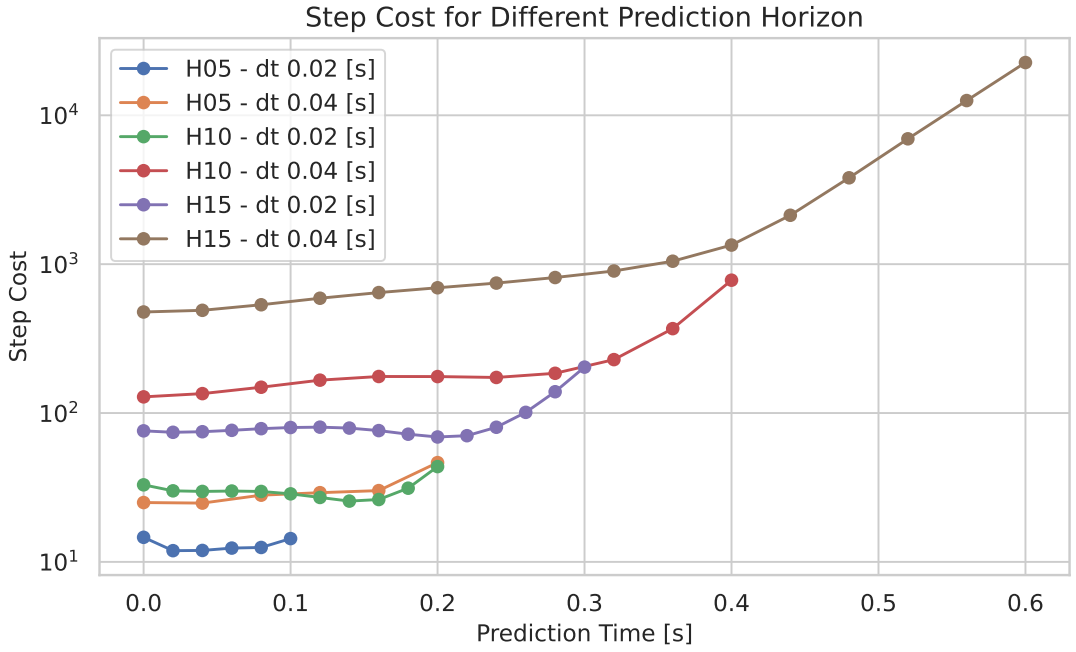


Figure 28: Effect of the prediction horizon on the rollout cost, with (*first-spline*) encoding, on a *lin-log* scale. All values are median values.

7.3.4 Effect of the Number of Samples on the Performances

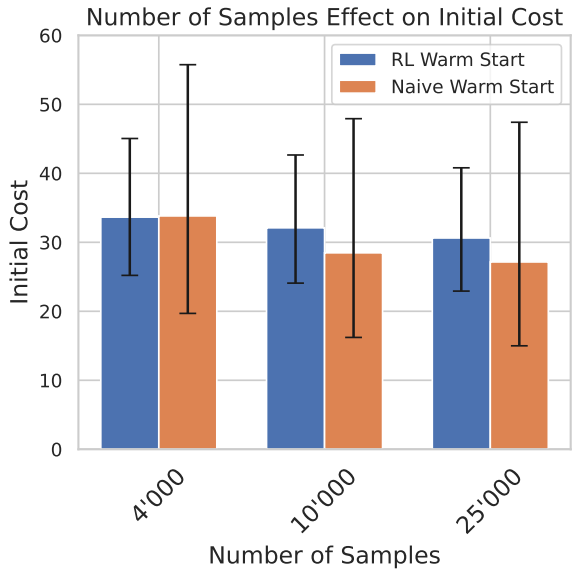


Figure 29: Comparison of the number of samples effect on the initial cost

Table 16 and figure 29 assess the influence of the number of samples on controller performance. The RL policy warm start makes the controller less sensitive to the number of samples. Variance is lower, and the solution is always relatively good. This approach definitely has a positive effect.

The number of samples increases controller performance, but this effect is small. In the remainder of this project, 10'000 samples will be used.

	Number of samples	RL Reward		Initial Cost		Rollout Cost	
RL warm starting	4000	2.348	+0.06%	33.64	+4.82%	333.5	+9.94%
	10000	2.347		32.10		303.3	
	25000	2.348	+0.06%	30.62	-4.59%	278.4	-8.21%
Naive warm starting	4000	2.252	-0.40%	33.82	+18.79%	243.4	+25.71%
	10000	2.261		28.47		193.6	
	25000	2.269	+0.35%	27.15	-4.65%	172.6	-10.83%
RL baseline		2.365		86.45			

Table 16: Comparison of the number of samples on the performances of the sampling controller

7.3.5 Gait optimization

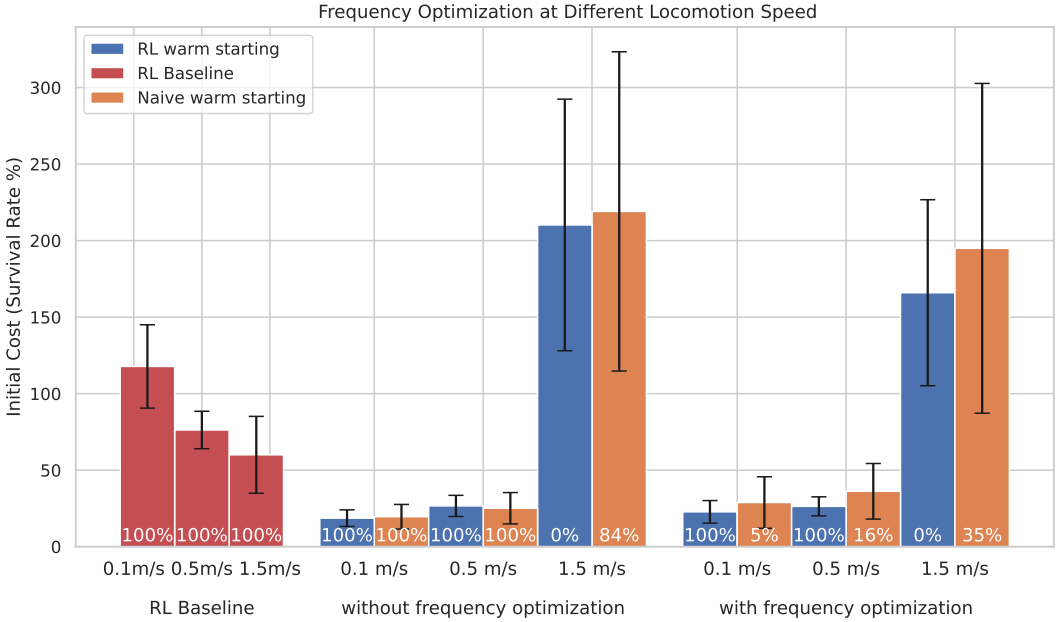


Figure 30: Effect of gait frequency optimization on the sampling controller

This chapter investigates the effect of optimizing the f and d gait parameters with the sampling MPC. To this end, the controllers were tested at three speeds, mainly forward on flat ground, without disturbances.

The results, illustrated in figure 30 and table 17, show that optimization is no longer effective at high speeds, and costs rise sharply. In addition, one observes that the imitation learning policy has not been able to generalize the trot of the base policy at high speed. It systematically falls when the speed leaves the distribution seen during learning. Moreover, one observes that it is difficult to fully grasp the behavior of an experiment given one metric.

In general, the optimization of gait parameters, while in some cases reducing costs, tends to destabilize the robot, which ends up falling. This can be explained by the fact that the MPC is short-sided and sees little influence from changing these parameters. Furthermore, the model does not consider leg dynamics and is, therefore, simplistic with respect to this effect. Finally, the gait parameters are not included in the cost function, so the optimization tends to make them vary a lot, which introduces instability.

Nevertheless, the author is convinced that these problems can be overcome but would require substantial adaptations. As a first step, the performance of the IL policy can certainly be improved. Part of the solution certainly lies in the learning procedure. On the one hand, training lasted thirty epochs, which is sufficient to no longer notice any progress on the quality of the data fit nor on the episodic reward.

However, policies trained by reinforcement learning are trained for much longer, and although episodic reward seems to have converged, they are still gaining precision and refining their locomotion ability. The difference between an RL policy at 1000, 3000, and 5000 training iterations can be seen visually, while the progress in reward is almost imperceptible. The same hypothesis seems likely for policies trained by imitation learning, which cannot generalize their behavior to situations outside training distributions.

Another part of the explanation certainly lies in the size of the dataset. It is relatively small because it has to fit entirely into the GPU's memory, along with all the data needed to run the simulation. Partitioning the dataset so that it can be stored in disk memory and thus increase its size would certainly have been beneficial. Nevertheless, the workload of this project did not allow this.

In a second step, the formulation of the sampling controller can be adapted to limit the effects of jittering. The variation of gait parameters with respect to a reference should be penalized. Other adaptations are also possible.

Finally, perhaps the MPC's prediction is simply not fine enough to optimize the nominal gait. Indeed, a variation of 0.1[Hz] is the smallest perceptible variation in its horizon. As a result, frequency optimization may be confined to responding to disturbances with greater frequency variations, as it has been done in [Turrisi et al. 2024].

	Gait optimization	RL Reward			Initial Cost			Survival Rate		
		$0.1\frac{m}{s}$	$0.5\frac{m}{s}$	$1.5\frac{m}{s}$	$0.1\frac{m}{s}$	$0.5\frac{m}{s}$	$1.5\frac{m}{s}$	$0.1\frac{m}{s}$	$0.5\frac{m}{s}$	$1.5\frac{m}{s}$
With RL warm starting	No	2.175	2.348	1.776	18.65	26.65	210.2	100%	100%	0%
	Frequency	2.177	2.347	1.781	22.78	26.37	165.9	100%	100%	0%
	Duty Cycle	2.063	2.219	1.794	28.74	39.20	200.7	100%	100%	22.6%
	Full	2.074	2.220	1.688	27.78	38.31	187.8	100%	100%	14.9%
Without warm starting	No	1.986	2.280	1.697	19.60	25.17	219.1	100%	100%	84.2%
	Frequency	1.304	1.706	1.695	28.88	36.20	194.9	5.1%	15.8%	35.1%
	Duty Cycle	1.173	1.603	1.595	34.47	24.32	61.25	5.0%	0%	0%
	Full	1.318	1.574	1.601	21.29	24.28	50.69	7.2%	0%	0%
RL Baseline		2.217	2.359	2.291	117.7	76.25	60.04	100%	100%	99.8%

Table 17: Effect of gait optimization on the sampling controller at different locomotion speed

7.3.6 Foothold optimization

The foothold optimization results were not conclusive and will not be presented in detail. The same issues as for gait optimization were observed with a lot of jittering despite the fact that their variation was penalized in the cost function.

Obtaining viable results would require adaptations that the time available did not allow. One adaptation would be to limit, for example, the periods when the MPC can optimize a touch-down position. The MPC should only optimize a touch-down position if it is actually used in its horizon and should no longer be able to modify it when the impact is imminent.

7.3.7 Conclusion

In this chapter, the results of the sampling controller were evaluated with a warm start from an RL policy. This approach greatly reduced the variance of the MPC sampling solution and slightly increased its overall quality. Moreover, one aspect that does not convey itself well in a table and graph analysis is that locomotion looks much more natural with the RL solution replacing the foothold generator and hand-designed gaits. The diversity of motion that can be generated has been significantly improved for the sampling controller.

In addition, it has come to light that the two approaches, RL and MPC, have different optimalities. This optimality is crafted into the reward and cost function of each of them. Even if their differences seem minimal, the author believes it has a significant impact on the results. The height tracking is a good

example: It has a large cost in the MPC, while it is of relative importance in the RL reward. Moreover, this tight constraint coming from the MPC hinders the movement from the RL policy. The approaches described in 5.3 could be an efficient way to mitigate this issue.

However, the results do not meet the author's expectations. The approach adopted extensively complexifies the control scheme, design procedure, and tuning for relatively lean benefits. Furthermore, although RL warm start slightly benefits the sampling controller, the contrary is not true. The RL solution performs better on its own. This was particularly true for the out-of-distribution speed test. One would have hoped that the sampling controller's inherent robustness would have helped the RL policy, but the opposite happened.

As mentioned, some of these limitations can be addressed, and the performance of the sampling controller can be significantly improved. In addition, end-to-end learning with the sampling controller in the loop (chapter 5.3) may also be beneficial. It would bring the optimality of the two controllers closer and maybe avoid the out-of-distribution effect. However, this may also impact training time.

It is now possible to answer the first research question: Is RL warm starting beneficial to the sampling controller? The answer is yes, RL warm starting benefits the sampling controller, but the opposite is not as it is. As stated, there are serious possibilities for improvement.

7.4 Effect of Different Policies on the Sampling Controller

In this chapter, we will evaluate the sampling controller warm started by the previously defined policies and attempt to answer the second research question. The climb task has been omitted from this analysis because it relies essentially on exteroceptive information, which unfortunately could not be integrated into the sampling controller during this project.

7.4.1 Performance on Simple Task

Warm start	RL Reward		Initial Cost		Survival Rate	
Base	526.5	-0.6%	26.7	-57.3%	100%	+0%
Speed	445.8	+6.4%	106.7	-69.7%	74%	-26%
Rough	388.0	-12.3%	203.7	-32.8%	96%	-2%
Base+Speed	364.1		91.9		84%	
Rough+Speed	429.2		94.9		93%	

Table 18: Comparison of sampling MPC performances with RL policies warm start on evaluation task. Percentage values represent the variation from the corresponding RL policy alone

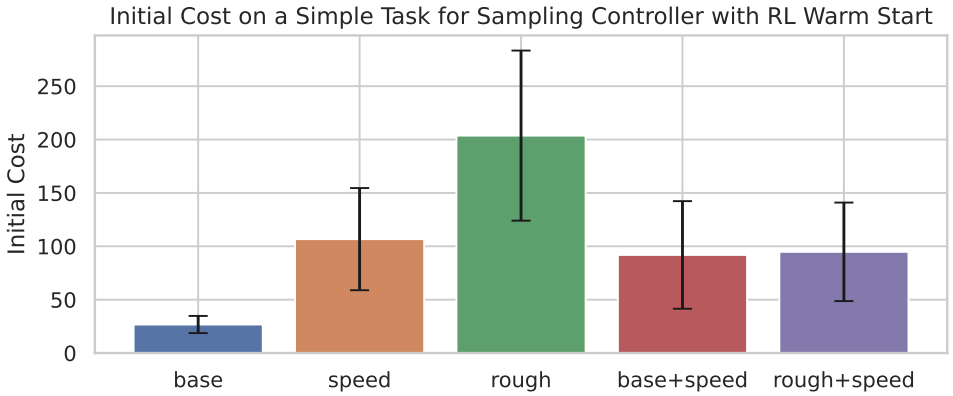


Figure 31: Comparison of the sampling controller with different warm start on a simple task (flat terrain, no disturbance, low-speed command)

The first comparison is on the simple task defined in chapter 7.1.2. This comparison allows the evaluation of the different controllers on a task, which is in distribution for all the policies. This provides a baseline for the analysis. Unsurprisingly, the warm start controller with the base policy performs best, corroborating with the conclusions drawn in chapter 7.1.2.

In addition, the sampling controller significantly lowers the median initial cost. However, good performance on this metric does not necessarily translate into better results. In general, the sampling controller tends to destabilize the RL policy. This can be assessed visually and seen in the lowered survival rate

Furthermore, the performances of the multipolicy warm start on this simple task are not particularly conclusive. The performances tend to be the average performance of the two policies combined rather than the best of the two.

7.4.2 Performance on Survival Task

The controller performance on the difficult task defined in chapter 7.1.3 has been evaluated and illustrated in figure 32. This time, there are several highly interesting things to note. Firstly, the survival rate of controllers with multiple warm starts was zero (and is therefore not shown). Secondly, the sampling controller has significantly improved the capabilities of the base policy, which has seen its survival probability increase by 38%. It reached the best survival rate without seeing any disturbances during

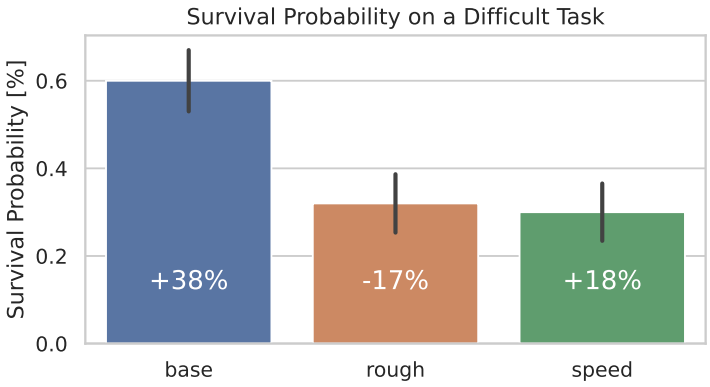


Figure 32: Comparison of the sampling controller survival probability on a difficult task, percentage represent variation from the corresponding RL policy alone

training and outperformed every other policy in this project. Nonetheless, the conclusions are unclear when we look at the results as a whole.

The hypothesis proposed is that the relatively fast trot of the base and speed policy are well suited to the sampling controller and its tuning. On the other hand, the rough policy has a very slow trot and keeps its legs up as much as possible, as if to delay the moment of deciding where to put its foot. This seems to be beneficial alone, as it makes it more resistant to disturbance. However, it does not work well with the MPC. The horizon may be too short for the MPC to see any contact, and it may simply believe it is falling, ultimately disrupting the system.

7.4.3 Performance at Different Speed

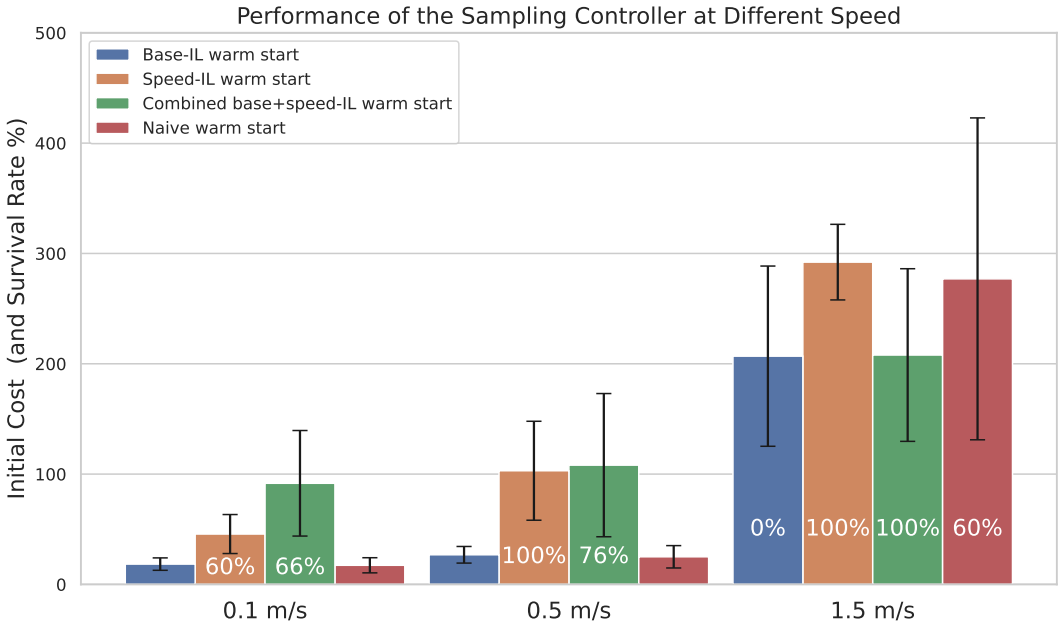


Figure 33: Comparison of the survival rate and initial cost for different warm start at three speeds

The performance of MPC sampling is evaluated at three different speeds: base policy, speed policy, and both at the same time. The effect one would have wished occurs at maximum speed. The two policies help each other. The base policy lowers the cost, while the speed policy maintains the maximum survival rate. However, this effect is only observed at this speed. In the other two cases, this coexistence

is detrimental.

This can be explained by the fact that, in the absence of significant differences in the optimality of one or the other, they conflict, and the solution chosen oscillates from one to the other. This behavior can be seen in figure 34, illustrating from which of the two policies the optimal solution comes. Note that there is a small amount of temporal consistency, although the solution can switch between the two policies up to 10 times per second (i.e., 1 time out of 10). This lack of consistency disturbs the policies and reduces performance. It is easy to imagine that there are two acceptable positions for the foot, but it is hard to believe that oscillating between these two choices is a good solution. Indeed, the sampling controller lacks a mechanism to ensure a good coexistence between optima. One could imagine a penalty for switching from one to the other, ensuring truly beneficial transitions and greater temporal coherence.

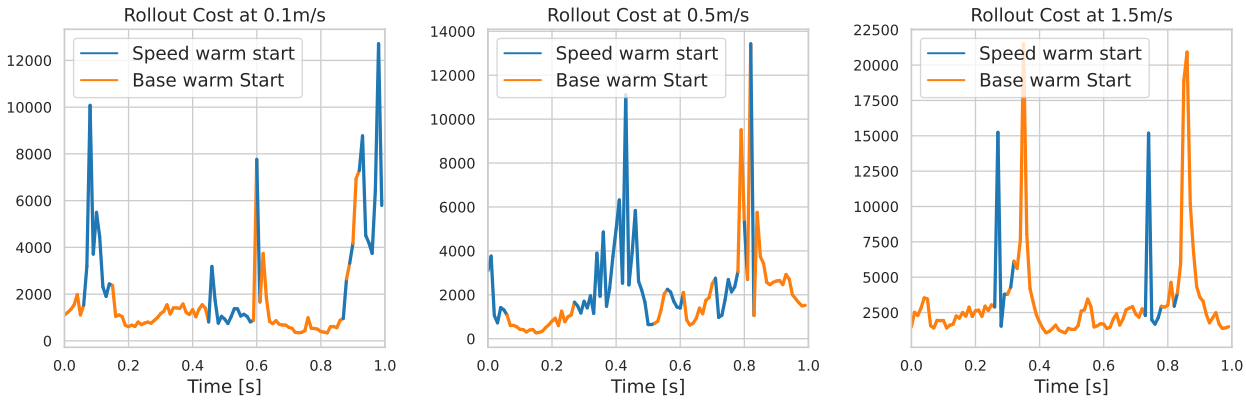


Figure 34: Example of rollout cost arbitration between the base and speed policy across 3 different speeds

7.4.4 Conclusion

This chapter discussed how different reinforcement learning policies perform when combined with the sampling controller. Their relationship is complex. It may be beneficial in some cases, but in others, it may not. For example, the sampling controller disturbs the base policy at high speed (table 17) but helps it greatly when there are strong disturbances (figure 32). Furthermore, some types of behavior are more suitable than others. The parameters and formulation of the MPC certainly play a major role in this process.

These results allow us to nuance the answer given to the first research question in the previous chapter. Yes, an RL policy's warm start of the sampling controller can, in some cases, benefit both the MPC and the RL policy. However, this requires specific conditions and does not benefit all types of policy.

To answer the second research question: RQ2 Can Multiple Warm Starting Be Beneficial? As it stands, multiple warm starts are detrimental to overall performance. There are some cases where it may be beneficial, but these are very limited. In addition, both policies and sampling controllers are sensitive to settings, and a single setting for all is certainly not appropriate. Nevertheless, arbitration techniques to ensure better policy cohabitation would certainly be beneficial. The answer given is no; multiple warm starts do not seem to be beneficial. Nevertheless, there is still room for improvement that could change the answer.

8 Conclusion

In this ambitious project, we have explored the integration of reinforcement learning with a sampling model predictive controller to enhance the capabilities of robotic locomotion. It was divided into three distinct parts

First of all, through a systematic approach, we developed and trained four RL policies, each tailored for a specific task: base, rough, speed, and climb. The initial results demonstrated the effectiveness of these policies. They overcame each task with a visual elegance that is unfortunately poorly conveyed in a report made of graphs and tables. Unfortunately, this observation is slightly tarnished by the out-of-tune policies, which could have performed better.

Secondly, we adapted the policies we obtained to the sampling controller's action space. We used imitation learning and adapted the DAgger algorithm to our particular problem. A detailed analysis of the influence of the various parameters was carried out, showing a priori excellent results. However, the author's intuition is that, despite the excellent numbers, these policies have not inherited all the robustness of their respective experts. Indeed, the size of the dataset was relatively small due to technical constraints, and the influence of excessively long training on their ability to generalize behaviors outside their distribution was not evaluated.

Then, the loop was closed by assembling these policies together with the sampling model predictive controller. The influence of numerous parameters was quantified, validating our approach. It was then possible to answer the two research questions. The results were mitigated by the extent to which they are sensitive to implementation details. It is certainly possible to do much better in every aspect. Nonetheless, we realize that this three-stage framework, all of which have to be tuned, is highly complex. One has to wonder whether this approach is really worth it, given the simplicity and results achieved by the RL alone.

Reducing the difference in optimality between the two problems is a very promising source of improvement. Indeed, training a policy with the cost of the sampling controller in its reward avoids the problem of imitation learning while reconciling the two optimality. Nevertheless, the MPC cost function may be too simplistic to train an RL policy effectively. Moreover, we have seen that our sampling controller is still too basic to optimize the gait parameters properly, and further implementations could remedy this.

Remerciements

Avec ce travail, je conclus définitivement mes études, et c'est une page de ma vie qui se tourne. Je tiens tout d'abord à remercier le professeur Ijspeert pour sa supervision, mais surtout pour m'avoir donné l'opportunité de réaliser mon projet en Italie. Je souhaite ensuite remercier le Dr Semini, également pour sa supervision, mais surtout pour m'avoir accepté au sein de son laboratoire. C'était un projet qui me tenait à cœur.

Ora devo naturalmente ringraziare Giulio per la passione che trasmette al suo lavoro e per l'aiuto che mi ha dato, molto più di quanto potessi sperare. Vorrei anche ringraziare tutti i colleghi del DLS che mi hanno accolto e con i quali l'esperienza è stata piacevole. In particolare, vorrei ringraziare Angelo, che è stato il primo a integrarmi, fin dall'intervista.

Vient ensuite, au tour de mes amis et de ma famille qui m'ont accompagné et soutenu tout au long de mon parcours. Mes amis du collège avec qui je suis venu à l'université. Ceux de toujours et ceux que je me suis fait en cours de route. Les compagnons avec qui j'ai partagé des projets, des bancs de classe ou des cafés. Les membres de l'association pour laquelle on s'est investis plus que de raison. Mes colocataires pour leur soutien quotidien. Ma mère qui a été fière de moi dès le départ. Finalement, j'ai une pensée particulière pour mon père, à qui j'aurais souhaité pouvoir montrer mon travail.

Appendix A Cubic Spline Parameters

In this appendix, cubic hermitian spline will be first defined, followed by a method for finding best fitting spline parameters, given a dataset. The notation presented in *Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo* from [Howell et al. 2022] will be used. However adaptation specific to the author's problem will be made.

A.1 Cubic Hermite Splines

A spline is a piecewise defined polynomial function. The cubic Hermite spline is spline where each piece is a cubic polynomial in Hermite form. The spline value and first derivative is defined at the control point, which ensure function continuity and first derivative continuity. In the most general case, one can define the hermitian spline made of P cubic polynomial s as :

- $x_{-1:P+1}$: a sequence of monotonically increasing values
- $\theta_{-1:P+1}$: a set of spline parameters (or control points) associated to $x_{-1:P+1}$
- $y = \Big\{ s(x, (x_{i-1:i+2}, \theta_{i-1:i+2})), x \in [x_i, x_{i+1}[$: A piece-wise function defined for $x \in [x_0, x_P[$

The hermitian cubic polynomial is defined as :

$$y = s(x, (x_{i-1:i+2}, \theta_{i-1:i+2})), \quad x \in [x_i, x_{i+1}[\tag{A.1.1}$$

and returns :

$$s = \alpha \cdot \theta_i + \beta \cdot \phi_i + \gamma \cdot \theta_{i+1} + \delta \cdot \phi_{i+1} \tag{A.1.2}$$

with :

$$q = \frac{x - x_i}{x_{i+1} - x_i} \tag{A.1.3}$$

$$\phi_i = \frac{1}{2}\left(\frac{\theta_{i+1} - \theta_i}{x_{i+1} - x_i} + \frac{\theta_i - \theta_{i-1}}{x_i - x_{i-1}}\right) \tag{A.1.4}$$

$$\phi_{i+1} = \frac{1}{2}\left(\frac{\theta_{i+2} - \theta_{i+1}}{x_{i+2} - x_{i+1}} + \frac{\theta_{i+1} - \theta_i}{x_{i+1} - x_i}\right) \tag{A.1.5}$$

$$\alpha = 2q^3 - 3q^2 + 1 \tag{A.1.6}$$

$$\beta = (q^3 - 2q^2 + q) \cdot (x_{i+1} - x_i) \tag{A.1.7}$$

$$\gamma = -2q^3 + 3q^2 \tag{A.1.8}$$

$$\delta = (q^3 - q^2) \cdot (x_{i+1} - x_i) \tag{A.1.9}$$

$$\tag{A.1.10}$$

This completely defines the Hermite spline in the most general case. It has several desirable properties. First, as previously mentioned, the function and its first derivative are continuous, ensuring a certain level of smoothness. Second, bounding the spline parameters mostly bound the spline [Howell et al. 2022]. Finally, shifting and scaling the spline parameters will also transform the spline itself, which is crucial when changing the reference frame.

One would have noticed that for P cubic polynomials, $P + 2$ spline parameters are required, and that the first and last interval aren't defined (ie. $[x_{-1}, x_0[$ and $[x_p, x_{p+1}]$). However, given the problem considered in this project, a few assumption can be made :

- A single cubic polynomial would be considered for the spline, thus $i = 0$
- The spline parameters are evenly spaced by $\Delta x = 1$
- Finally the spline is define interval on the unit interval, thus $x_0 = 0, x_1 = 1$

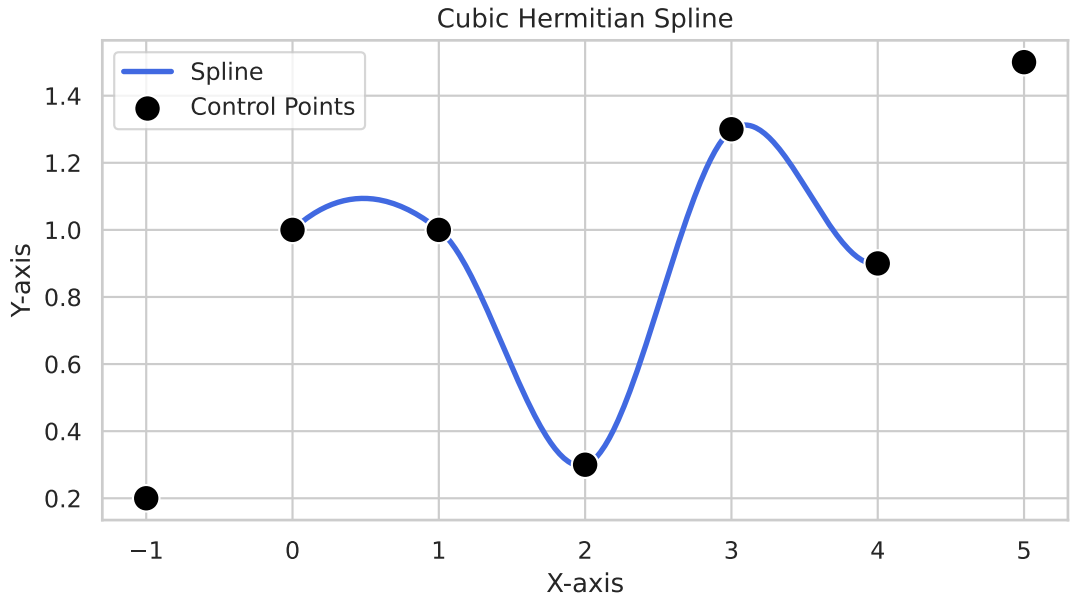


Figure 35: An example of an hermitian spline, defined over the interval $x \in [0, 4]$

This simplify the problem and it can be rewritten as :

$$q = x \tag{A.1.11}$$
$$\phi_i = \frac{1}{2}(\theta_{i+1} - \theta_{i-1}) \tag{A.1.12}$$
$$\phi_{i+1} = \frac{1}{2}(\theta_{i+2} - \theta_i) \tag{A.1.13}$$
$$\alpha = 2q^3 - 3q^2 + 1 \tag{A.1.14}$$
$$\beta = q^3 - 2q^2 + q \tag{A.1.15}$$
$$\gamma = -2q^3 + 3q^2 \tag{A.1.16}$$
$$\delta = q^3 - q^2 \tag{A.1.17}$$
$$\tag{A.1.18}$$

Which ultimately gives :

$$\begin{aligned} s(x) = &(-\frac{1}{2}\theta_{-1} + \frac{3}{2}\theta_0 - \frac{3}{2}\theta_1 + \frac{1}{2}\theta_2) \cdot x^3 \\ &+ (\theta_{-1} - \frac{5}{2}\theta_0 + 2\theta_1 - \frac{1}{2}\theta_2) \cdot x^2 \\ &+ (-\frac{1}{2}\theta_{-1} + \frac{1}{2}\theta_1) \cdot x \\ &+ \theta_0 \end{aligned} \tag{A.1.19}$$

A.2 Determining Spline Parameters

Given the appealing properties of the cubic hermitian spline, they are often used for data interpolation. It is an efficient way to reduce the complexity of the problem, where the information of numerous data points can be embedded in fewer simpler parameters. Nonetheless, a method to determine these parameters is required, which is the aim of this paragraph. The formulation of the hermitian spline has been simplified to correspond to this project problem. One could then identify the parameters to express the cubic polynomial in its canonical form as :

$$f(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d \tag{A.2.1}$$

with :

$$a = -\frac{1}{2}\theta_{-1} + \frac{3}{2}\theta_0 - \frac{3}{2}\theta_1 + \frac{1}{2}\theta_2 \tag{A.2.2}$$

$$b = \theta_{-1} - \frac{5}{2}\theta_0 + 2\theta_1 - \frac{1}{2}\theta_2 \tag{A.2.3}$$

$$c = -\frac{1}{2}\theta_{-1} + \frac{1}{2}\theta_1 \tag{A.2.4}$$

$$d = \theta_0 \tag{A.2.5}$$

$$\tag{A.2.6}$$

Let's consider a dataset (x, y) of N pairs of data points (x_i, y_i). The objective is to find the parameters a, b, c and d that minimize the error E between $f(x_i)$ and y_i, which is a cubic interpolation problem. A common choice for the error function E is the *least square error*, which will be used in this project. The problem can be formulated as :

$$\min_{a,b,c,d} \sum_{i=0}^{N} -1\big(y_i - (ax_i^3 + bx_i^2 + cx_i + d)\big)^2 \tag{A.2.7}$$

The problem can be modified to become linear and a closed form solution of this problem exists. First, let's construct the *design matrix* X

- Let $X = [x^3\ x^2\ x\ 1]$ be the $N \times 4$ design matrix, with $x \in R^N$
- Let y be the vector of N observation, with $y \in R^N$
- Let $\beta = [a, b, c, d]^T$ be the vector of coefficients

The problem can be written as a *linear least-squares problem*[13].

$$\min_{\beta} ||X\beta - y||^2 \tag{A.2.8}$$

The exact solution is when the derivative of the error E is equal to 0, which can be formulated as :

$$\beta = (X^T X)^{-1} X^T y \tag{A.2.9}$$

which is known as the normal equation for (A.2.8) [Nocedal et al. 2006, p. 250]. Equation (A.2.9) is a set of linear equations and can be efficiently solved by a computer, with for example *beta = torch.linalg.solve(XtX, Xty)*[Paszke et al. 2019].

A.3 Determining Spline Parameters with constraints

The above formulation gives flexibility in the problem and allows for example to incorporate constraints in the problem. Considering this project problem where the datapoint to be fitted are forces along a time horizon, with the first force eventually applied to the system. It could be desirable that the the first data point y_0 fitting not only is minimize the error, but is also exact (ie. $f(x_0) = y_0$).Given the spline formulation (A.1.19), this implies having the constraint $\theta_0 = y_0$

The polynomial function can be written as :

$$f(x) = ax^3 + bx^2 + cy + y_0 \tag{A.3.1}$$

In order to integrate this constraint, the problem must be reformulated :

- Let $X' = [x'^3\ x'^2\ x']$ be the $N - 1 \times 3$ design matrix, with $x' = [x_1, ..., x_{N-1}]^T$
- Let y' be the vector of $N - 1$ observation, with $y = [y_1 - y_0, ..., y_{N-1} - y_0]^T$

[13] For a comprehensive discussion on linear least-squares problems, see Chapter 10.2 of *Numerical Optimization* by Nocedal and Wright [Nocedal et al. 2006].

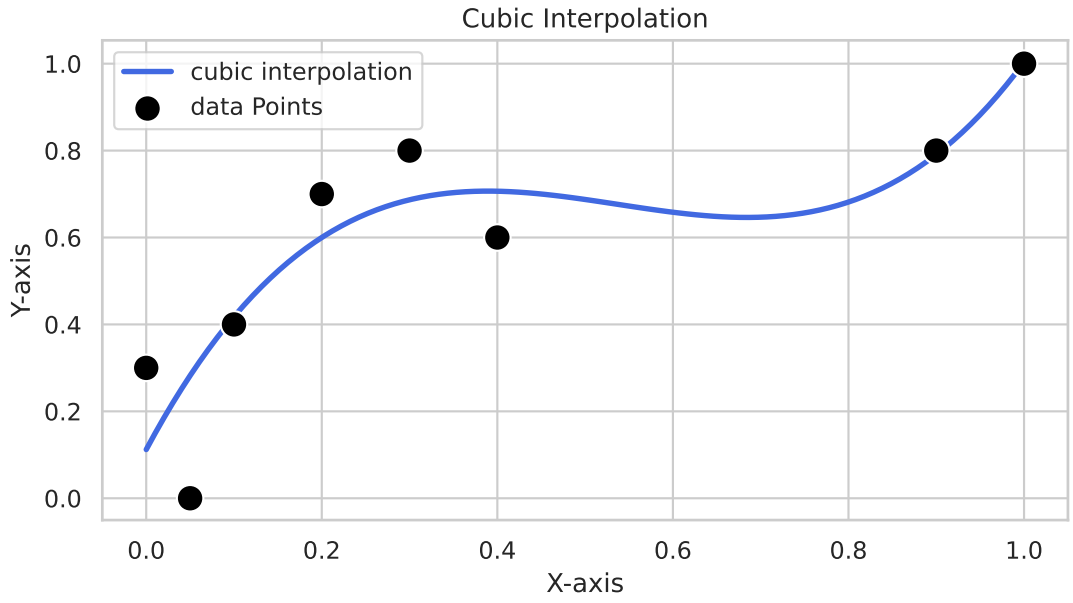


Figure 36: An example of an cubic interpolation

- Let $\beta = [a, b, c]^T$ be the vector of coefficients

Which ultimately leads to :

$$\beta' = (X'^T X')^{-1} X'^T y' \tag{A.3.2}$$

That can be solved as previoulsy giving $\beta = [\beta', y_0] = [a, b, c, y_0]$.

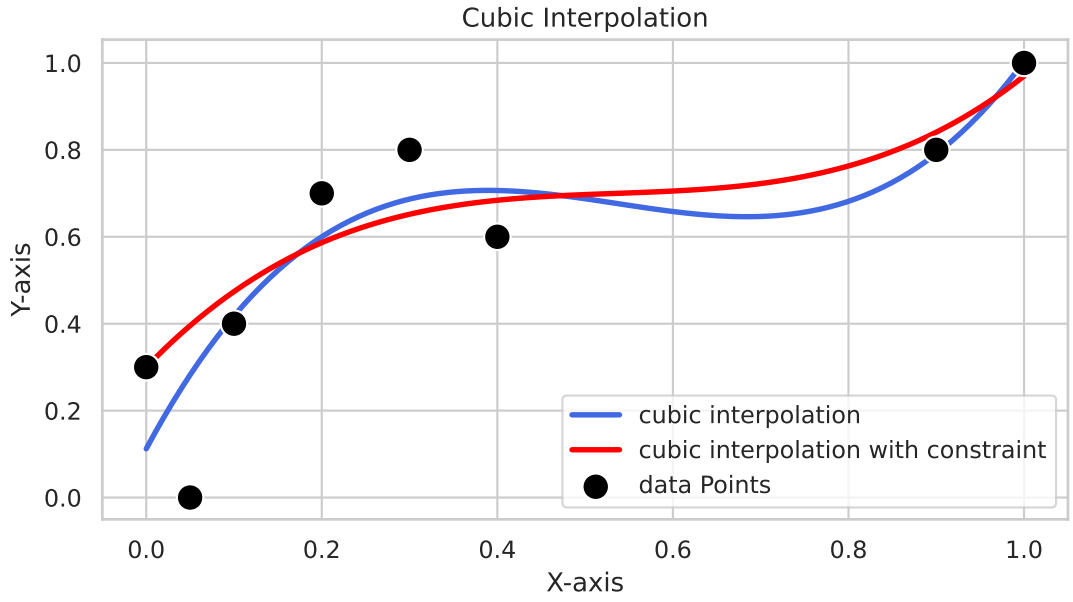


Figure 37: An example of an cubic interpolation, with constraint that the interpolation is exact for y_0

Appendix B Swing Trajectory generator

In this Appendix, the mathematical basis for the swing trajectory generator will be presented. As previously stated : a swing trajectory defines the path of the foot during its swing or airborne phase. It is characterized by key parameters including the lift-off and touch down position as well as the step height.

The objective of the swing trajectory generator is to produce a smooth trajectory that mimics well the natural locomotion. This trajectory will be later tracked by a *swing trajectory controller* (chapter 2.2.5). For smooth movement, the trajectory must be defined in position, velocity and acceleration. The approach adopted in this project leverages the properties of Bézier curves to compute the first and second derivative obtaining the aforementioned three trajectories. The required control points are defined heuristically to ensure the desired position, velocity and acceleration profiles

B.1 Cubic Bézier Curve

This project uses cubic Bézier curve B, which is the lowest order that can provide smooth and non-null velocity and acceleration. The cubic Bézier curve is defined by four control points (cp_1, cp_2, cp_3, cp_4) and by the equation :

$$\begin{aligned} B\colon [0,1] &\longrightarrow \mathbb{R} \\ t &\longmapsto cp_0(1-t)^3 + 3cp_1t(1-t)^2 + 3cp_2t^2(1-t) + cp_3t^3 \end{aligned} \tag{B.1.1}$$

The Bézier curve pass through cp_1 and cp_4 but generally not by cp_2 and cp_3 as they determine the direction and the rate of change of the curve.

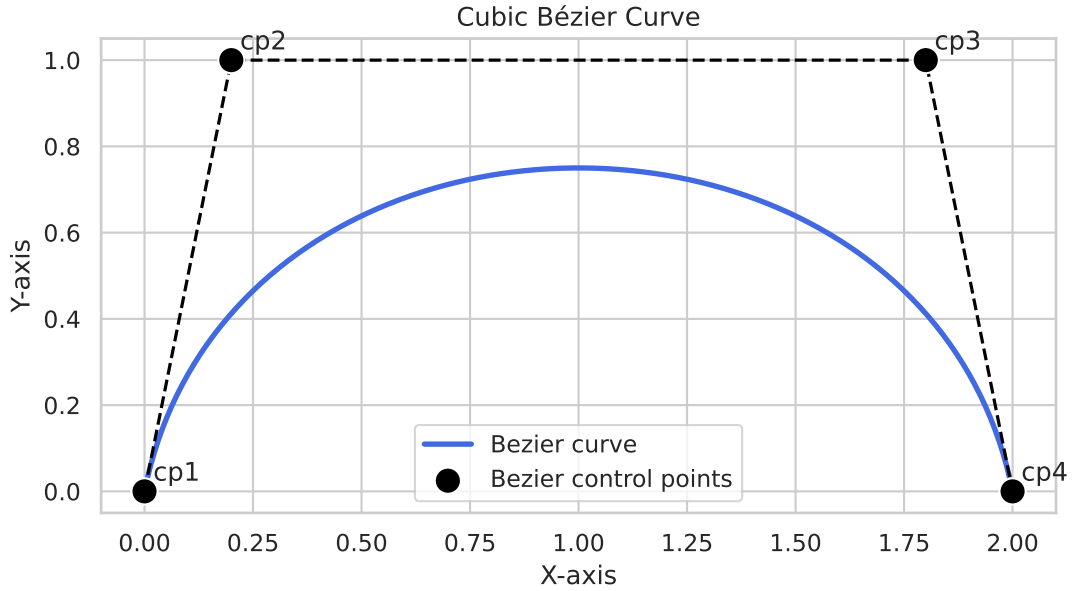


Figure 38: An example of an cubic Bézier curve

It is interesting to note, that B is defined as $B(t) = x$ and not as $B(x) = y$, because the variable t is used to generate the curve. Thus if one needs to a function that maps x to y, one would need to generate two sets of control points and two Bézier curves ($B(t, cp_x) = x$ and $B(t, cp_y) = y$), as illustrated in figure 38.

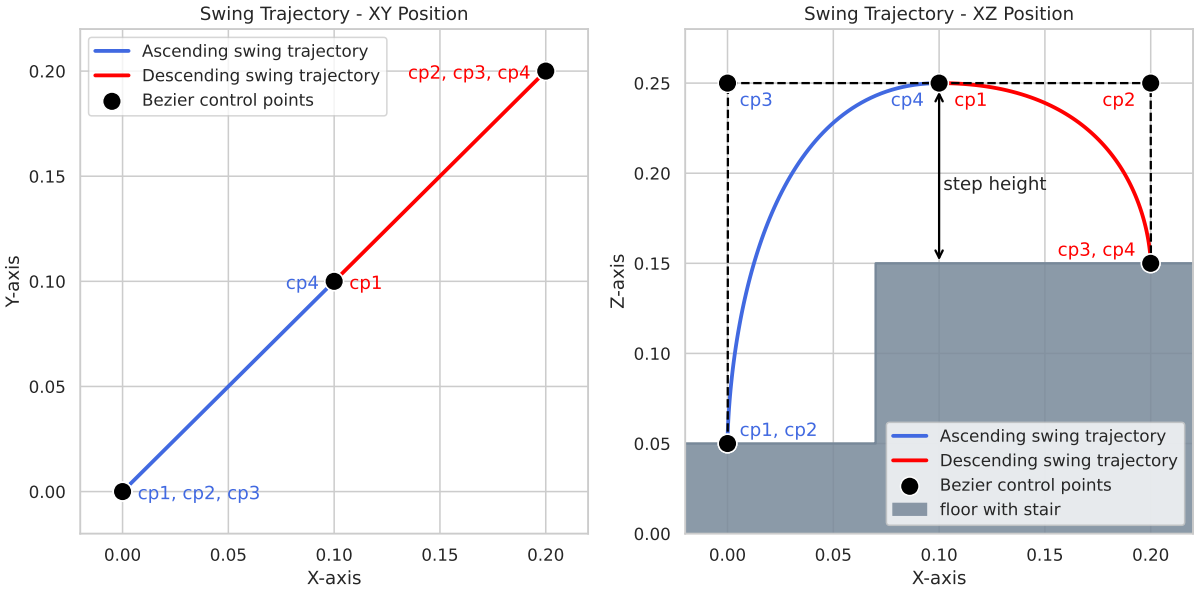
This comes with a nice property because the variable $t \in [0, 1]$ can be associated to time in the trajectory by a simple scale by the swing period. cp_1 and cp_4 are associated to the time at the start and end of the trajectory respectively. cp_2 and cp_3 don't have time associated and can be heuristically defined depending on the property wished.

B.2 Swing Trajectory Generator

The swing trajectory generator will leverage the Bézier curves and property of time associated to the trajectory. Two cubic Bézier curves will be used, one for the ascending phase, the other for the descending phase, and will both last half of the swing period. This can be defined as :

$$f(t) = \begin{cases} B(t_a, cp_a), & t \in \left[0, \frac{T_s}{2}\right] \\ B(t_d, cp_d), & t \in \left[\frac{T_s}{2}, T_s\right] \end{cases} \tag{B.2.1}$$

With T_s the swing period, t_a and t_d the scaled time $\in [0, 1]$, and cp_a and cp_d, the control point in the ascending and descending phase, respectively. The control points are defined heuristically with the foot lift-off position, the desired foot touch down postion, the terrain height and a desired step height. They can be retrieved in the figure 39.



(a) XY position of the swing trajectory

(b) XZ position of the swing trajectory

Figure 39: Swing trajectory in space of step

Figure 39 illustrates the swing trajectory in 3D space. However, the trajectory needs to be controlled with respect to time. It has been shown earlier that, thanks to appropriate scaling, a Bézier curve can be interpreted as a trajectory in time also. The first and second derivatives can easily be derived from $B(t)$ defined in equation (B.1.1). Thanks to the cubic Bézier curve property, the first derivative is continuous.[14]

[14]Please note that the time in these plot isn't scaled and thus time, velocity and acceleration don't have proper units.

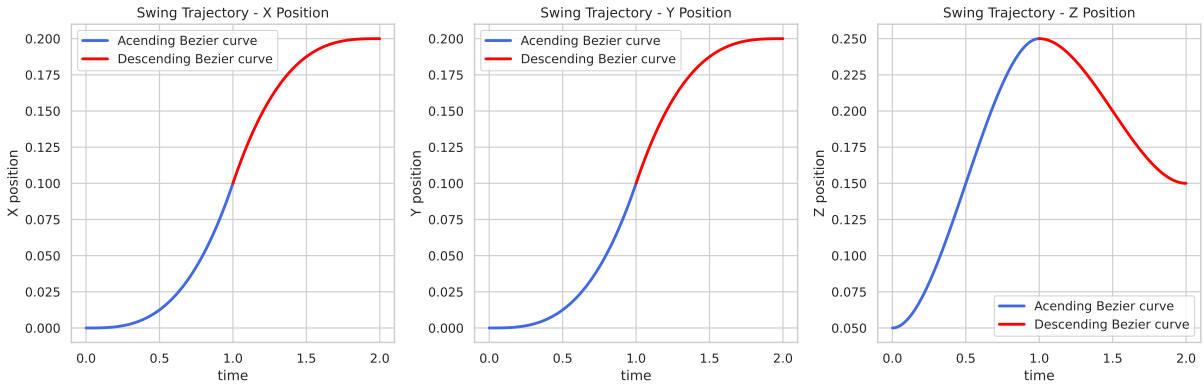


Figure 40: The swing trajectory position across time

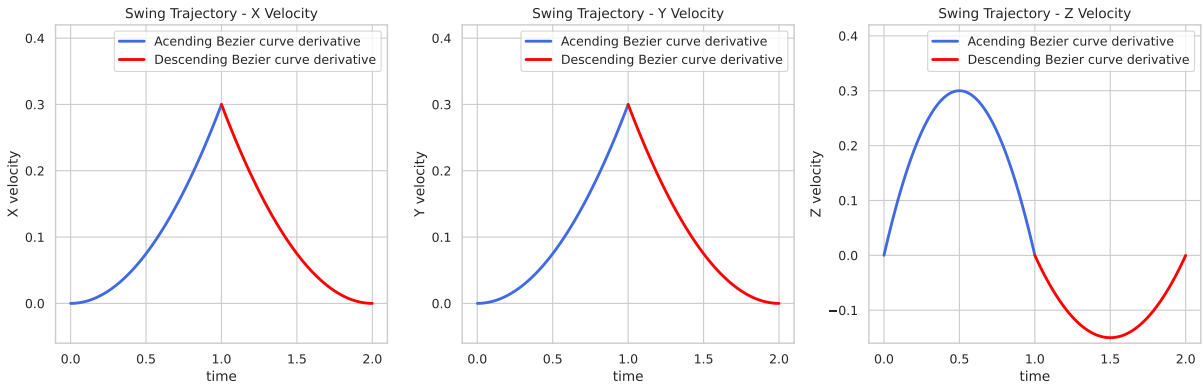


Figure 41: The swing trajectory velocity across time

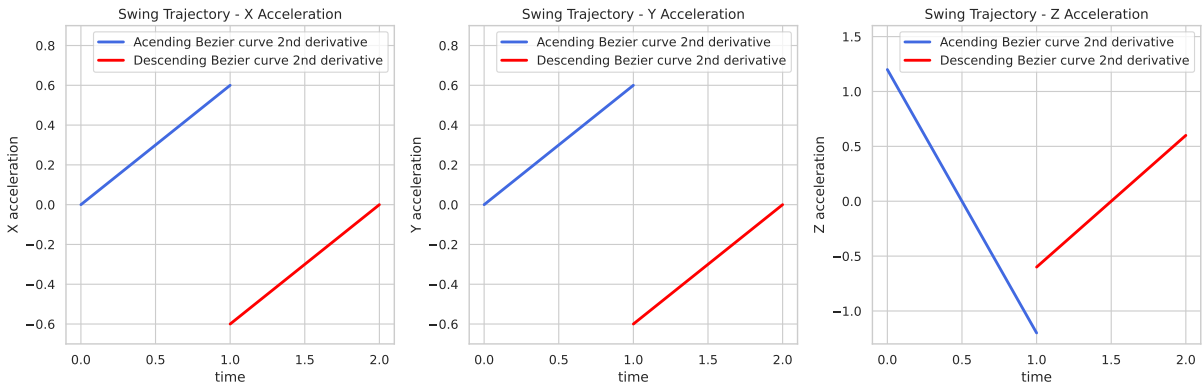


Figure 42: The swing trajectory acceleration across time

Appendix C Single Rigid Body Dynamics

The Single Rigid Body Dynamics (SRBD) model is a simple model used to simulate the robot dynamics. The model neglects the inertia of the legs (mass-less legs), and the entire robot is considered as a single rigid body. These assumptions reduce the model accuracy but allow a very simple model formulation that can be computed rapidly, which is essential for any model predictive controller (MPC). The assumptions make sense for quadruped robot that tends to minimize leg inertia and concentrate their mass in their body. Extensive information on the single rigid body dynamics can be found in *Model Predictive Control with Environment Adaptation for Legged Locomotion* from [Rathod et al. 2021].

C.1 Model Definition

The *original* model describes the dynamics of the robot center of mass position and orientation and their derivatives. It is described by a state $x \in \mathbb{R}^{12}$, and consider as input ground reaction forces at the feet $u \in \mathbb{R}^{4\times 3}$. However, in this project, we augmented the model state with the feet position and the input with desired feet touch down position. Moreover, to take into consideration a robot moving with legs in swing, a contact status is used to deal with discrete ground reaction forces.

Let's define the variable and frame used in this chapter :

- $\mathcal{W}$: The world inertial frame
- $\mathcal{B}$: The base frame attached to the robot's base (center of mass).
- m : The robot mass
- ${}_{\mathcal{W}}\boldsymbol{p}_c$: The robot center of mass position $\in \mathbb{R}^3$ in world frame $\mathcal{W}$.
- ${}_{\mathcal{W}}\boldsymbol{v}_c$: The robot center of mass velocity $\in \mathbb{R}^3$ in world frame$\mathcal{W}$.
- $\boldsymbol{\Phi}_c$: The robot base orientation with respect to world frame $\in \mathbb{R}^3$ as Euler angles in the XYZ convention [Diebel 2006] (ie. $\boldsymbol{\Phi} = (\phi, \theta, \psi) = (roll, pitch, yaw)$).
- ${}_{\mathcal{B}}\boldsymbol{\omega}_c$: The robot center of mass angular velocity $\in \mathbb{R}^3$ in base frame $\mathcal{B}$.
- ${}_{\mathcal{W}}\boldsymbol{p}_f$: The robot feet position $\in \mathbb{R}^{12}$ in world frame $\mathcal{W}$.
- ${}_{\mathcal{W}}\boldsymbol{p}_{f,ref}$: The desired feet touch down position $\in \mathbb{R}^{12}$ in world frame $\mathcal{W}$.
- ${}_{\mathcal{W}}\boldsymbol{F}_f$: The feet ground reaction forces $\in \mathbb{R}^{12}$ in world frame $\mathcal{W}$.
- ${}^{\mathcal{W}}_{\mathcal{B}}R$: The rotation matrix $\in SO(3)$ from base frame $\mathcal{B}$ to world frame $\mathcal{W}$.
- $\boldsymbol{\delta}_f$: Contact status of the feet with the ground $\in \{0, 1\}^4$.
- ${}_{\mathcal{B}}\boldsymbol{I}_c$: Center of mass interia tensor $\in \mathbb{R}^{3\times 3}$ in base frame $\mathcal{B}$.

The notation introduced here is complex but eliminates any ambiguity. Going forward, we will use a more relaxed notation and not always specify the reference frame. Unless explicitly stated otherwise, a variable without subscript will refer to the variable presented here.

One can then define the model state x and input u :

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_c \\ \boldsymbol{v}_c \\ \boldsymbol{\Phi} \\ \boldsymbol{\omega} \\ \boldsymbol{p}_f \end{bmatrix}, \qquad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{\delta} \\ \boldsymbol{p}_{f,ref} \\ \boldsymbol{F} \end{bmatrix} \tag{C.1.1}$$

The angular dynamics are defined in the base frame $\mathcal{B}$, which yields a constant inertia tensor ${}_{B}I$ and simplify the angular dynamics equations. The equations of dynamics for a single rigid body can then be expressed as :

$$\begin{aligned} m\dot{\boldsymbol{v}}_c &= m\boldsymbol{g} + \sum_{i=1}^{4} \delta_i \boldsymbol{F}_i \\ \boldsymbol{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times I\boldsymbol{\omega} &= \sum_{i=1}^{4} \delta_i (\boldsymbol{p}_c - \boldsymbol{p}_{f,i}) \times ({}^{\mathcal{W}}_{\mathcal{B}}R \cdot {}_{\mathcal{W}}\boldsymbol{F}_i) \end{aligned} \tag{C.1.2}$$

The robot dynamics described in equation (C.1.2) can be express in the continuous state space representation for a reduced state $\boldsymbol{x}'$ as :

$$\begin{aligned} \dot{\boldsymbol{x}}' &= f(\boldsymbol{x}', \boldsymbol{u}') \\ \begin{bmatrix} \dot{\boldsymbol{p}}_c \\ \dot{\boldsymbol{v}}_c \\ \dot{\boldsymbol{\Phi}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} &= \begin{bmatrix} \dot{\boldsymbol{v}}_c \\ \boldsymbol{g} + \frac{1}{m}\sum_{i=1}^{4} \delta_i \boldsymbol{F}_i \\ \left(\boldsymbol{E}'(\boldsymbol{\Phi})\right)^{-1} \boldsymbol{\omega} \\ \boldsymbol{I}^{-1}\left[\left(\sum_{i=1}^{4} \delta_i (\boldsymbol{p}_c - \boldsymbol{p}_{f,i}) \times (R \cdot \boldsymbol{F}_i)\right) - (\boldsymbol{\omega} \times I\boldsymbol{\omega})\right] \end{bmatrix} \end{aligned} \tag{C.1.3}$$

Which can then be discretized by any need necessary. However, the feet position evolution is not trivial to describe in continuous time. This project leverage the discrete time implementation of the problem and the fact that dynamics of swing leg is neglected to describe the feet position evolution in time. When the foot is in stance, the position don't change, assuming no slipping. When the foot is in swing, it's position evolve but it has no influence on the rest of the dynamics. Thus, it can already be freely set to the desired touch down position $\boldsymbol{p}_{f,ref}$. This assumption is reasonable as long as the reference is feasible and can be effectively tracked by the *swing controller*. The feet dynamics can be described as :

$$\boldsymbol{p}_{f,t+1} = (\boldsymbol{\delta} \cdot \boldsymbol{p}_{f,t}) + ((1 - \boldsymbol{\delta}) \cdot \boldsymbol{p}_{f,ref,t}) \tag{C.1.4}$$

C.2 Euler Angles and Euler Rates

A comprehensive and definitive explanation of the Euler angles and Euler rates can be found in the excellent work *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors* from [Diebel 2006]. In this chapter, only the relevant equations will be presented. The Euler angles convention used in this project is the XYZ convention, commonly referred to as *Cardan angles* or *nautical angles*. The Euler angles $\Phi = (\phi, \theta, \psi)$ can be interpreted respectively as *roll*, *pitch* and *yaw*.

Given the Euler angles $\boldsymbol{\Phi}$, one can compute the rotation matrix ${}^{\mathcal{W}}_{\mathcal{B}}R$ in closed form [Diebel 2006, p. 11]:

$$\begin{aligned} {}^{\mathcal{W}}_{\mathcal{B}}R(\phi, \theta, \psi) &= R_x(\phi) R_y(\theta) R_z(\psi) \\ &= \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\theta s_\phi \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\theta c_\phi \end{bmatrix} \end{aligned} \tag{C.2.1}$$

The derivative of the Euler angles Φ are the Euler rates $\dot{\Phi}$. The Euler rates can be related to the body angular velocity ω through the Euler angle rates matrix E. Their relationship can be described by [Diebel 2006, p. 9]:

$$\begin{aligned} {}_{\mathcal{W}}\boldsymbol{\omega} &= \boldsymbol{E}(\boldsymbol{\Phi}) \cdot \dot{\boldsymbol{\Phi}} \\ {}_{\mathcal{B}}\boldsymbol{\omega} &= \boldsymbol{E}'(\boldsymbol{\Phi}) \cdot \dot{\boldsymbol{\Phi}} \end{aligned} \tag{C.2.2}$$

With $\boldsymbol{E}'$ the Euler angle rates matrix conjugate. The Single Rigid Body Model requires the inverse of the Euler angle rates matrix conjugate for which a closed-form solution exists [Diebel 2006, p. 12] :

$$\left(\boldsymbol{E}'(\boldsymbol{\Phi})\right)^{-1} = \frac{1}{c_\theta} \begin{bmatrix} c_\theta & s_\phi s_\theta & c_\phi s_\theta \\ 0 & c_\phi c_\theta & -s_\phi c_\theta \\ 0 & s_\phi & c_\phi \end{bmatrix} \tag{C.2.3}$$

Lastly, a detail worth mentioning is the singularities of the Euler angles. In the chosen XYZ convention, there is a single singularity: when the pitch θ approaches $\pm 90°$ and the robot is vertical. This problem is well known [Diebel 2006, p. 13][Ding et al. 2021, p. 7] and referred to as *gimbal lock*. A common solution to address this issue is to switch the representation of the Euler angles as the singularity is approached. For instance, one could switch from the XYZ convention to the ZXZ convention. This approach would require minimal implementation changes, as the model could remain otherwise identical. However, since this project focuses on locomotion, the robot's pitch remains fairly flat, and such an implementation has not been deemed necessary.

Appendix D Soft Squared Exponential Kernel

In this project, a novel kernel function is defined : the soft squared exponential kernel. The main idea behind is to relax the constraint on certain directions. Let's remind the *standard* squared exponential kernel [15]:

$$\begin{aligned} f : \mathbb{R}^{N\times 2} &\longrightarrow]0,1] \\ (x,x') &\longmapsto e^{-\frac{||x-x'||^2}{2\sigma^2}} \end{aligned} \tag{D.0.1}$$

This kernel rewards similarity (as euclidean distance) between observation x and target x'. It has the property of being always positive and maximal if and only if $x = x'$. Divergence is exponentially penalized and it can be tuned with the hyperparameter σ. It is typically used in Reinforcement learning to design reward functions, for example, in this project, the velocity tracking function. Tolerance with respect to the tracking can be achieved with the hyperparameter δ.

However, the climbing task defined in this project required more flexibility that couldn't be achieved with the standard squared exponential kernel. Indeed, climbing a stair is a challenging obstacle, and the robot may not be able to clear the task at the desired velocity, which will lead to a fall. Especially if the desired velocity profile is a constant profile. However, it may be able to complete the task with a different velocity profile.

Intuitively, one can imagine that the robot would prepare the step by doing smaller step as it approach the obstacle, reducing its velocity. Then it may clear the obstacle in one bigger movement, augmenting it's velocity. Crafting such a velocity profile is certainly difficult and artificially imposes a behavior on how the robot should clear the task, which is not desirable.

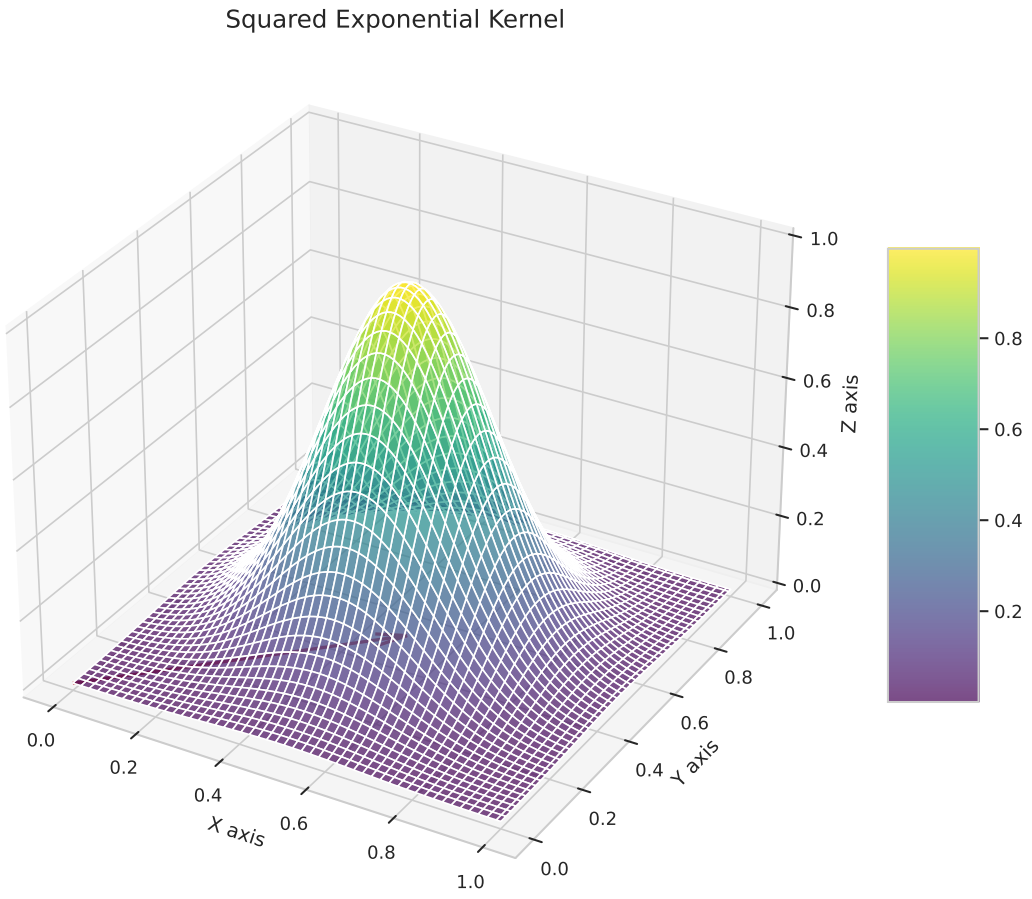


Figure 43: An example of the squared exponential kernel, with $(x, y)' = (0.5, 0.5)$ and $\sigma = 0.25$

Another approach would be to give some slack in the velocity tracking on difficult obstacles in order

[15] Please note, that implementation details and names in *IsaacLab* may differ

for the model to learn an optimal velocity profile, while being able to clear the obstacle and moving in the desired direction. This is the motivation behind the soft squared exponential kernel.

Slack on the tracking velocity can somehow be achieved with a larger δ hyperparameter. However, this is not very desirable since it gives slack on every direction, while one want to give slack only in the direction of the obstacle. The tracking on the direction perpendicular to the commanded velocity must remain stiff. In addition, the overall tracking is worse, no matter the difficulty of the obstacles. Finally, only perfect tracking will provide maximum reward.

The initial step is to identify the directions of interest: the direction of the commanded speed and the direction perpendicular to it. Let's define the basis $\mathcal{O}$ with the composed of the unit vectors $(\hat{e}_x, \hat{e}_y)$. The speed command in the XY plane is expressed as $(v_{ref,x}, v_{ref,y})$ in the basis $\mathcal{O}$. Let's define a new basis $\mathcal{O}'$ composed of the unit vectors $(\hat{e}_{x'}, \hat{e}_{y'})$, with $\hat{e}_{x'} \parallel \vec{v}_{ref}$ the first axis parallel to the commanded speed.

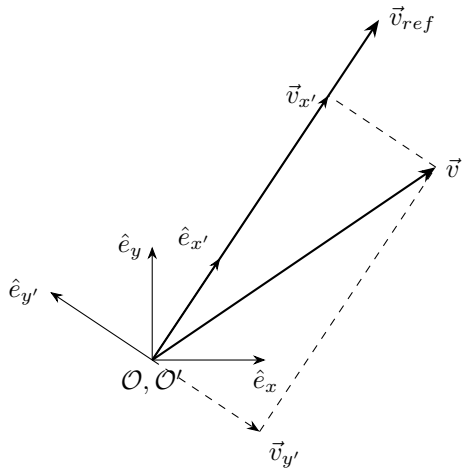


Figure 44: Example of projection and decomposition of $\vec{v}$ in the new basis $\mathcal{O}'$

$$\begin{aligned} \hat{e}_{x'} &= \frac{\vec{v}_{ref}}{||\vec{v}_{ref}||} \\ \hat{e}_{y'} &= \begin{bmatrix} -(\hat{e}_{x'})_y \\ (\hat{e}_{x'})_x \end{bmatrix} \end{aligned} \tag{D.0.2}$$

With this, we have the two directions of interest :

- $\hat{e}_{x'}$: The direction of the commanded speed, in which we want to provide some slack according to the obstacle difficulty.
- $\hat{e}_{y'}$: The direction perpendicular to the commanded speed, in which we want to remain stiff.

The second step is to compute the divergence between the commanded speed $\vec{v}_{ref}$ and the current speed $\vec{v}$ in the new basis $\mathcal{O}'$, to obtain, namely the *forward velocity error* $e_{/\!/}$ and the *lateral velocity error* $e_\perp$.

$$\begin{aligned} \text{Forward velocity error :} \quad e_{/\!/} &= ||\vec{v}_{ref}|| - \frac{\vec{v}_{ref} \cdot \vec{v}}{||\vec{v}_{ref}||} \\ \text{Lateral velocity error :} \quad e_\perp &= -\frac{\vec{v}_{ref} \times \vec{v}}{\vec{v}_{ref}} \end{aligned} \tag{D.0.3}$$

Now that the errors are separated in two components of interest, slack can be added to the tracking direction. For this purpose, a piece-wise affine function $h(x)$ has been crafted in equation (D.0.4):

$$h(x) = \begin{cases} x\ , & x \in]0, +\infty[\\ 0\ , & x \in\ [-\alpha, 0] \\ x + \alpha\ , & x \in\ ]-\infty, -\alpha[\end{cases} \qquad \text{(D.0.4)}$$

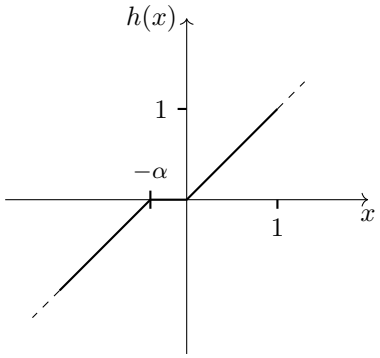


Figure 45: Piece-wise affine function to add slack

Everything can be bring together to craft the soft squared exponential kernel :

$$\begin{aligned} f : \mathbb{R}^2 &\longrightarrow]0, 1] \\ (e_{/\!/}, e_\perp) &\longmapsto \exp\left(-\frac{h(e_{/\!/})^2 + e_\perp^2}{2\sigma^2}\right) \end{aligned} \qquad \text{(D.0.5)}$$

The slack can be tuned by the hyperparameter α, which is function of the terrain difficulty in this project. As the stairs become harder, the robot as more slack in forward velocity tracking, and hopefully is able to clear the obstacles.

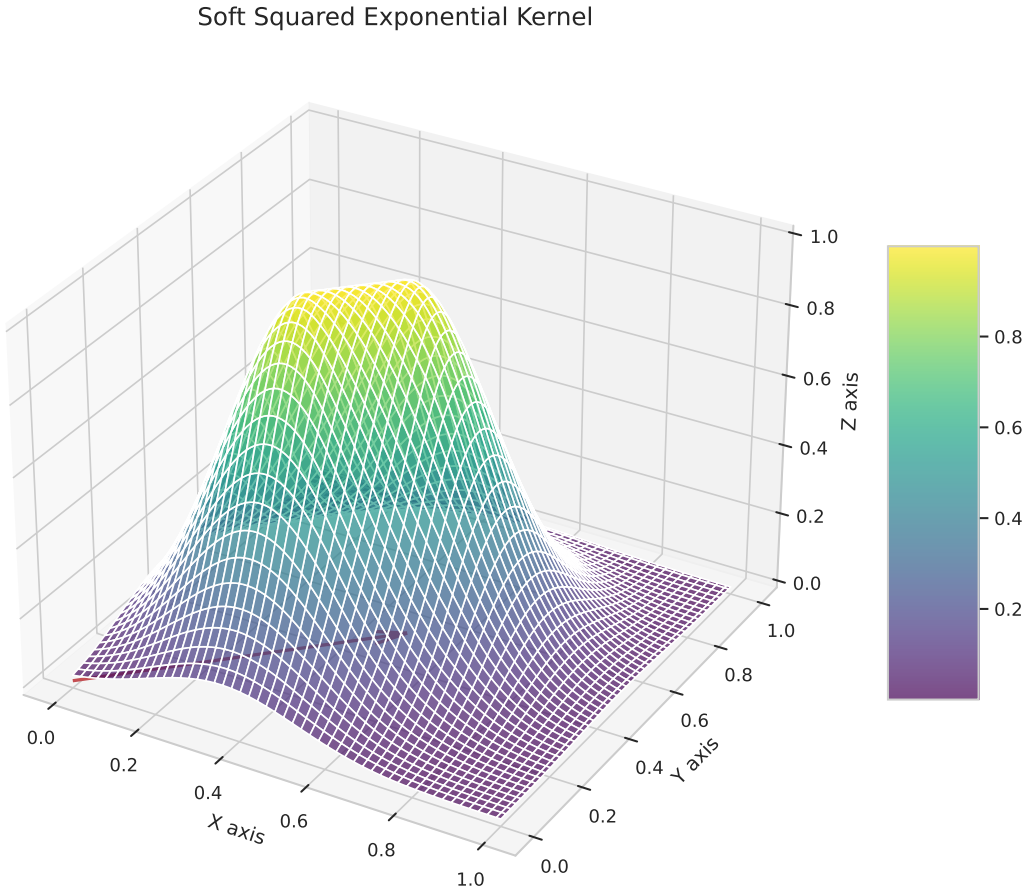


Figure 46: An example of the soft squared exponential kernel, with $\vec{v}_{ref} = (0.5, 0.5)$, $\sigma = 0.25$ and $\alpha = 0.2$

References

Alessandro, De Luca (2011). *Cartesian Control*. Accessed: 2024-08-01.

Asmar, Dylan M. et al. (2023). *Model Predictive Optimized Path Integral Strategies*.

Aude, Billard, Mirrazavi Sina, and Figueroa Nadia (2022). *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. Cambridge, MA: The MIT Press. Chap. 10.

Barasuol, Victor et al. (2013). "A reactive controller framework for quadrupedal locomotion on challenging terrain". In: *2013 IEEE International Conference on Robotics and Automation*, pp. 2554–2561.

Bill, Baxter (2002). *Fast Numerical Methods for Inverse Kinematics*. Accessed: 2024-08-01. URL: `https://billbaxter.com/courses/290/html/index.htm`.

Bradbury, James et al. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13.

Caron, Sébastien (2023). *Joint Torques and the Jacobian Transpose*. Accessed: 2024-08-01. URL: `https://scaron.info/blog/joint-torques-and-jacobian-transpose.html`.

Cheng, Xuxin et al. (2023). *Extreme Parkour with Legged Robots*.

Diebel, James (2006). "Representing Attitude : Euler Angles , Unit Quaternions , and Rotation Vectors". In.

Ding, Yanran et al. (Aug. 2021). "Representation-Free Model Predictive Control for Dynamic Motions in Quadrupeds". In: *IEEE Transactions on Robotics* 37.4, pp. 1154–1171. ISSN: 1941-0468.

Frison, Gianluca and Moritz Diehl (2020). *HPIPM: a high-performance quadratic programming framework for model predictive control*.

Fujimoto, Scott, Herke van Hoof, and David Meger (2018). *Addressing Function Approximation Error in Actor-Critic Methods*.

Gros, Sebastien and Zanon Mario (Feb. 2020). "Data-Driven Economic NMPC Using Reinforcement Learning". In: *IEEE Transactions on Automatic Control* 65.2, pp. 636–648. ISSN: 2334-3303.

Gros, Sebastien et al. (Sept. 2016). "From linear to nonlinear MPC: bridging the gap via the real-time iteration". In: *International Journal of Control* 93, pp. 1–19.

Haarnoja, Tuomas et al. (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*.

Hildebrand, Milton (Dec. 1989). "The Quadrupedal Gaits of Vertebrates: The timing of leg movements relates to balance, body shape, agility, speed, and energy expenditure". In: *BioScience* 39.11, pp. 766–775. ISSN: 0006-3568.

Hoeller, David and Nikita Rudin (2023a). *RSL_RL: A Modular Framework for Reinforcement Learning in Robotics*. Version 1.0.0. GitHub repository. Robotic Systems Lab (RSL), ETH Zurich.

Hoeller, David et al. (2023b). "ANYmal Parkour: Learning Agile Navigation for Quadrupedal Robots". In: *CoRR* abs/2306.14874.

Howell, Taylor et al. (2022). *Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo*.

Kusumoto, Raphael et al. (2019). "Informed Information Theoretic Model Predictive Control". In: *2019 International Conference on Robotics and Automation (ICRA)*, pp. 2047–2053.

Latypov, Oleg (2019). *A Comprehensive Guide to Proximal Policy Optimization (PPO) in AI*. Accessed: 2024-08-04.

Lee, Joonho et al. (Oct. 2020). "Learning quadrupedal locomotion over challenging terrain". In: *Science Robotics* 5.47. ISSN: 2470-9476.

Lembono, Teguh Santoso et al. (2020). *Learning How to Walk: Warm-starting Optimal Control Solver with Memory of Motion*.

Liu, Shuijing (2019). *Introduction to DAgger*. University of Trento.

Martinsen, Andreas B., Anastasios M. Lekkas, and Sébastien Gros (2022). "Reinforcement learning-based NMPC for tracking control of ASVs: Theory and experiments". In: *Control Engineering Practice* 120, p. 105024. ISSN: 0967-0661.

Mastalli, Carlos et al. (2022). *Agile Maneuvers in Legged Robots: a Predictive Control Approach*.

Mittal, Mayank et al. (2023). "Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments". In: *IEEE Robotics and Automation Letters* 8.6, pp. 3740–3747.

Neunert, Michael et al. (July 2018). "Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds". In: *IEEE Robotics and Automation Letters* 3.3, pp. 1458–1465. ISSN: 2377-3774.

Nocedal, Jorge and Stephen Wright (2006). *Numerical Optimization*. Chapter: Linear Least-Square Problems. Springer. Chap. 10.2, pp. 371–382.

Omar, Shafeef et al. (2023). *SafeSteps: Learning Safer Footstep Planning Policies for Legged Robots via Model-Based Priors*.

Orin, David, Ambarish Goswami, and Sung-Hee Lee (Oct. 2013). "Centroidal dynamics of a humanoid robot". In: *Autonomous Robots* 35.

Pandala, Abhishek et al. (2022). "Robust Predictive Control for Quadrupedal Locomotion: Learning to Close the Gap Between Reduced- and Full-Order Models". In: *IEEE Robotics and Automation Letters* 7.3, pp. 6622–6629.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035.

Pinneri, Cristina et al. (2020). *Sample-efficient Cross-Entropy Method for Real-time Planning*.

Rathod, Niraj et al. (2021). "Mobility-enhanced MPC for Legged Locomotion on Rough Terrain". In: *CoRR* abs/2105.05998.

Rawlings, J.B., D.Q. Mayne, and M. Diehl (2017). *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing. ISBN: 9780975937730.

Ross, Stéphane, Geoffrey J. Gordon, and J. Andrew Bagnell (2010). "No-Regret Reductions for Imitation Learning and Structured Prediction". In: *CoRR* abs/1011.0686.

Schulman, John et al. (2017a). *Proximal Policy Optimization Algorithms*.

Schulman, John et al. (2017b). *Trust Region Policy Optimization*.

Semini, Claudio et al. (Aug. 2011). "Design of HyQ -A hydraulically and electrically actuated quadruped robot". In: *Proceedings of the Institution of Mechanical Engineers. Part I: Journal of Systems and Control Engineering* 225, pp. 831–849.

Sutton, R.S. and A.G. Barto (1998). "Reinforcement Learning: An Introduction". In: *IEEE Transactions on Neural Networks* 9.5, pp. 1054–1054.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 5026–5033.

Towers, Mark et al. (Mar. 2023). *Gymnasium*.

Turrisi, Giulio et al. (2024). *On the Benefits of GPU Sample-Based Stochastic Predictive Controllers for Legged Locomotion*.

Yang, Yuxiang et al. (2021). *Fast and Efficient Locomotion via Learned Gait Transitions*.

Zanon, Mario and s Sebastien Gro (Aug. 2021). "Safe Reinforcement Learning Using Robust MPC". In: *IEEE Transactions on Automatic Control* 66.8, pp. 3638–3652. ISSN: 2334-3303.

Zeng, Xuanqi et al. (2019). "Leg Trajectory Planning for Quadruped Robots with High-Speed Trot Gait". In: *Applied Sciences* 9.7, p. 1508.