POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Closing the Reality Gap by Estimating Dynamic Residuals with Gaussian Processes

CLÉMENT GARANCINI
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Juillet 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Closing the Reality Gap by Estimating Dynamic Residuals with Gaussian Processes

présenté par **Clément GARANCINI**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

DEDICATION

À tous mes amis du labos,
vous me manquerez...

ACKNOWLEDGEMENTS

The work that has been accomplished during this master's degree would not have been possible without help. So I would like to thank Giovanni Beltrame for his supervision during those two years and all the member of MIST Lab. More specifically, I am grateful to Hassan Ali for the provided mentorship and guidance, even an ocean apart. I would also like to thank deeply Yann Boutelier, Soma Karthik and Maeva Guerrier for their help and friendship. I would also like to sincerely thank Lola le Breton and Charles de Malefette, my roommates, for their support outside of the lab, always ready to listen to me talking about robots. Cheers!

RÉSUMÉ

L'apprentissage par renforcement (Reinforcement Learning, RL) s'est récemment imposé comme un cadre puissant pour résoudre des problèmes complexes de prise de décision, faisant de lui un candidat prometteur pour des applications de contrôle de systèmes mécaniques réels. Le déploiement de ces agents de Reinforcement Learning (RL) dans des environnements réels restant difficile en raison des coûts élevés, des risques et du temps requis pour l'entraînement direct sur les systèmes physiques, l'utilisation de la simulation pour l'entraînement s'est imposé comme une l'approche standard, exploitant les capacités de calcul et de parallélisation des ordinateurs modernes pour développer des politiques ensuite transférées sur les systèmes réels. Cependant, malgré les progrès des simulateurs en matière de fidélité et des techniques de transfert des agents de la simulation vers la réalité, le déploiement des politiques RL sur des systèmes physiques reste limité par le réalité gap — c'est-à-dire le décalage entre le comportement simulé et le comportement réel - lorsque ces systèmes présentent des dynamiques complexes.

Ce mémoire explore une approche visant à combler cet écart entre simulateur et monde réel en améliorant le simulateur lui-même. Plus précisément, nous proposons une méthode reposant sur les processus Gaussiens (Gaussian Processes, GP) pour modéliser les résidus des dynamiques entre l'environnement simulé et le monde réel. En optimisant un GP à partir de données collectées dans le système réel et en l'ajoutant comme terme correctif dans la fonction de transition de la simulation, nous obtenons un simulateur enrichi capable de générer des trajectoires plus fidèles à la réalité. Cet environnement amélioré est ensuite utilisé pour entraîner de nouvelles politiques de RL plus robustes au transfert vers l'environnement cible et présentant de meilleure performance dans celui-ci.

Afin de valider l'efficacité de cette approche, nous avons réalisé des expériences en simulation et sur un robot dans le monde réel. Nos résultats montrent non seulement que l'ajout du modèle correctif au simulateur permet de générer des trajectoires plus réalistes mais aussi que les politiques entrainées sur ce simulateur atteignent des performances quasi-optimales une fois transférées dans l'environnement réel. Ce travail souligne l'intérêt de modèles probabilistes, peu gourmands en données, pour améliorer le transfert de simulation vers réalité des politiques de RL, participant ainsi à réduire l'écart entre simulation et monde réel dans le cadre de l'apprentissage par renforcement appliqué à la robotique.

ABSTRACT

Reinforcement Learning has recently emerged as a powerful framework for solving complex decision-making problems and made it a promising candidate for solving real-world tasks such as robotic control. However, the deployment of RL in real-world scenarios remains challenging due to the high cost, risk, and time associated with training directly on physical systems. To avoid these constraint, simulation-based training has become a standard approach, leveraging fast, parallelizable environments to develop policies that are then transferred to real hardware. Despite advancements in simulation fidelity and sim-to-real transfer techniques, the deployment of RL policies to hardware systems with highly complex dynamics continues to face the persistent issue of the reality gap — the discrepancy between simulated and real-world behaviors.

This thesis investigates a real-to-sim approach to bridge this gap by enhancing the simulator itself. Specifically, we propose a method based on Gaussian Processes to model the residual dynamics between the simulated and real environments. By learning a GP from real-world interaction data and integrating it into the simulation loop as an additive correction term, we produce an enhanced simulator that generates system behaviors closer to the one encountered in the real world. This improved simulation environment is then used to train new RL policies more robust to deployment in the target environment.

We validate the effectiveness of this approach with both simulation and physical experiments. Our results demonstrate that the Gaussian process enhancement of the environment enable generation of more realistic trajectories and the training of policies that achieve near-optimal performance in the target real-world environment. This work highlights the value of data-efficient, probabilistic modeling for improving Sim-to-Real transfer and contributes to closing the loop between simulation and real-world deployment in RL for robotic systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

RL	Reinforcement Learning
RBF	Radial Basis Function
GP	Gaussian Process
Sim-to-Real	Simulation to Reality
Real-to-Sim	Reality to Simulation
MDP	Markov Decision Process
ENVIA	Simulation Environment for AI-driven vehicles

CHAPTER 1 INTRODUCTION

This thesis presents the research that was done at the Making Innovative Space Technologies Laboratory (MISTLab) of Polytechnique Montréal and conducted in the context of a research master that took place between September 2023 and June 2025. The thesis is presented in the format of a thesis by article that compose the fourth (4) chapter and is currently under review for publication in a scientific journal.

To fully understand the impact this work sought to reach, we should begin by introducing the context in which it places itself along with some key concepts used to form both the problem formulation and the solution that are explored here. We will then see how those concepts interact in the problem formulation that we tried to tackle before seeing the methodology we established in order to solve this problem.

1.1 Context, definitions and key concepts

This work is part of an ongoing project, *ENVIA*, in collaboration with Flying Whales Quebec and Thales and shared between one postdoc and three students at MIST Lab. It is mandatory to detail this project in order to understand the motivations for this work along with some decisions that were made during the evolution of this work. Before entering in the details of this project, some key concepts and definitions must be established.

1.1.1 Reinforcement Learning

Reinforcement Learning (RL) is a paradigm of Machine Learning in which the task to solve is formulated as a of problem sequential decision making under uncertainty. It is composed of an environment and an agent that interact over discrete time steps. According to the state sent by the environment, the agent choose an action making the environment transitioning in the next state. A reward is sent to the agent to provide a signal for the "quality" of its decision along with the next state and so on.

Formally, the RL is formulated as Markov Decision Process (Markov Decision Process (MDP)) (S, A, P, R, γ) where:

- $\mathcal{S}$ is the set of possible states,
- $\mathcal{A}$ is the set of possible actions,

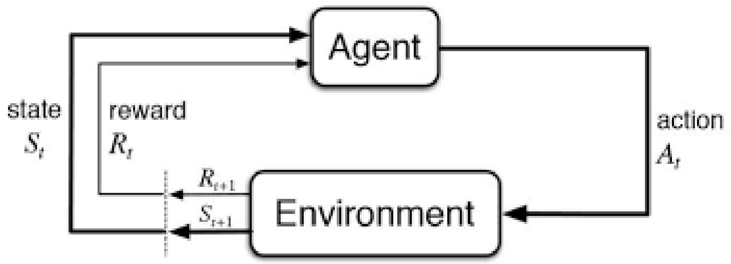


Figure 1.1 Schematic representation of the RL framework [1]

- $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the *transition function*, where $\mathcal{P}(\mathcal{S})$ denotes the set of probability distributions over $\mathcal{S}$. For each state-action pair (s, a), $P(\cdot|s, a)$ defines the distribution over next states,
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward function*, mapping the current state and action chosen to a scalar reward,
- $\gamma \in [0, 1)$ is the discount factor.

The agent decision function, called a *policy*, is the stochastic function $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ that maps the states to a distribution over the action space. The agent's goal is to find a policy π that maximizes the expected discounted return:

$$J(\pi) = \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right] \tag{1.1}$$

where the expectation is taken over trajectories generated by the policy π and the transition dynamics P. In this expression, the discount factor represent the tradeoff in importance of the immediate and long-term reward.

When this policy is parameterized by a set of parameters θ we denote it π_θ. The agent's objective can be formulated as learning the optimal policy π_θ^* that maximize the expected cumulative discounted reward:

$$\pi^* = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \begin{array}{l} a_t \sim \pi_\theta(\cdot|s_t), \\ s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t) \end{array} \right] \tag{1.2}$$

A central concept in RL is the *state-value function* $V^\pi(s)$, which quantifies the expected return when starting from state s and following the policy π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\middle| s_0 = s\right] \tag{1.3}$$

Since the transition and reward functions are often unknown in practice, the agent needs to find the optimal parameters θ^* by interacting with the environment. Through repeated episodes of trial-and-error, the agent estimates the value function and improves its policy using optimization algorithms that can go from dynamic programming to policy gradient techniques.

1.1.2 Reality Gap

A challenge that arise when trying to apply reinforcement learning to real-world systems is known as the *Reality Gap*, which refers to the discrepancy between simulated environments and the physical world. While simulations offer a large range of advantages for the training of RL agent such as safety, computational speed, and cost-effectiveness, they inevitably simplify or omit key aspects of real-world physics, dynamics, noise, and sensing. As a result, policies trained in simulation may fail when one try to deploye them on real hardware due to unmodeled phenomena.

Formally, one can consider two MDPs: the *simulated* MDP $\mathcal{M}_{\text{sim}} = (\mathcal{S}, \mathcal{A}, P_{\text{sim}}, R_{\text{sim}}, \gamma)$ and the *real-world* MDP $\mathcal{M}_{\text{real}} = (\mathcal{S}, \mathcal{A}, P_{\text{real}}, R_{\text{real}}, \gamma)$. The reality gap manifests when $P_{\text{sim}} \neq P_{\text{real}}$, leading to a mismatch in the optimal policy π^* derived from $\mathcal{M}_{\text{sim}}$ versus the one needed for $\mathcal{M}_{\text{real}}$. Addressing this issue requires either to reduce the discrepancy between P_{sim} and P_{real} or to train policies that are robust to the Simulation-to-Reality Simulation to Reality (Sim-to-Real) transfer. Techniques such as system identification or domain randomization can help to achieve one or the other of these goals.

1.1.3 Reality-to-Simulation

When trying to reduce the dynamic discrepancies that exists between a simulation and the real world, one can tune the current parameters of the simulator according to real world data and/or physic domain-specific knowledge. This is often done through system identification but can also be learned with Machine Learning techniques. On the other hand, one can try to learn new dynamics that can take the form of latent or explicit representations of real-world dynamics, creating a *world model* enabling more accurate simulation.

Aligning the transition functions of the simulation and the real world can, by producing more realistic trajectories, help reduce the reality gap and then the transferability of the policies learned in such simulated environments.

1.1.4 ENVIA Project

Now that we defined the overall context of this thesis, we can explain the specific context of the project the work presented here is part of.

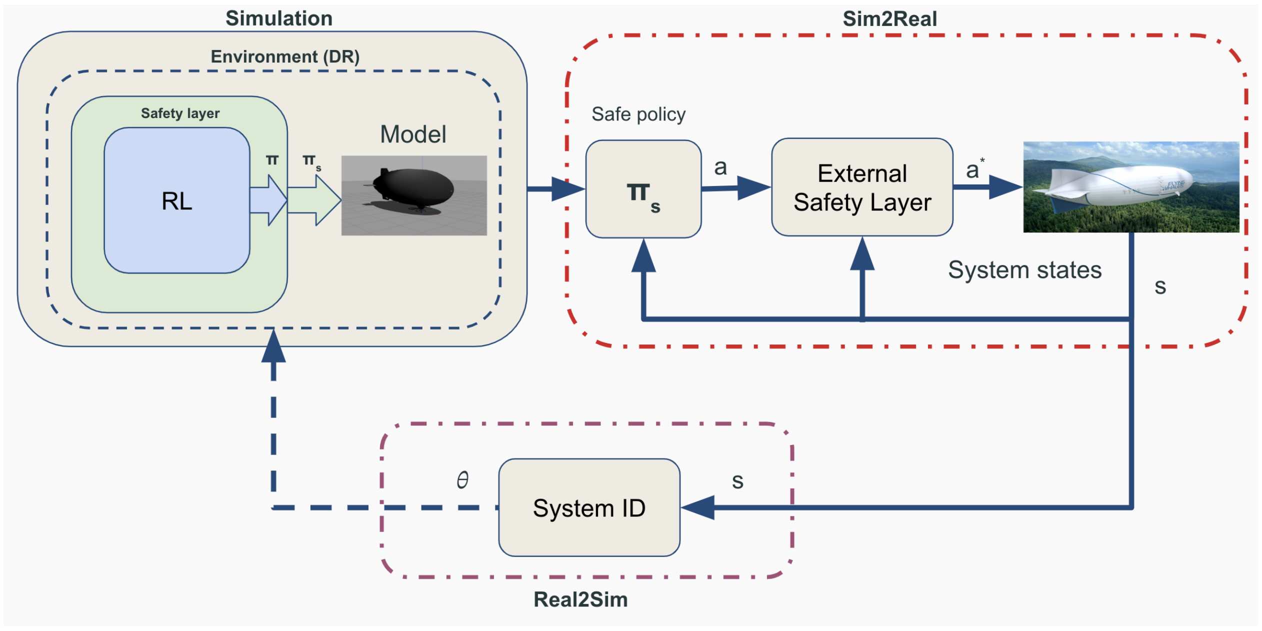


Figure 1.2 ENVIA project framework

The Simulation Environment for AI-driven vehicles (Simulation Environment for AI-driven vehicles (ENVIA)) project is a collaboration project that was conducted jointly by the MIST-Lab and the company Flying Whales Québec. The global objective of this project is the production of safe RL policies for blimp control through a full-loop real-2-sim-2-real training framework. This generic framework is presented in the figure 1.2.

The work presented here can be seen as a part of this global project and aims to tackle the Real-to-Sim part of the aforementioned graph.

1.2 Problem elements

While simulations are essential for scalable and safe training, they are inherently approximations of the true physical processes. This mismatch in dynamics, the reality gap, hinders the direct transfer of RL policies trained in simulation to the real world. In particular, the dynamics $P_{\text{sim}}(s'|s,a)$ used in simulation often diverge from the true dynamics $P_{\text{real}}(s'|s,a)$, resulting in suboptimal or unstable behaviors. Because the simulation dynamics are not only inaccurate in general, but also a simplification of the real world, the policies trained in simulation often find ways to "hack" the system leading to catastrophic behaviors in the real world.

1.2.1 Reality gap

The problem element of the Sim-to-Real transfer in presence of a reality gap rise the problem of measuring this gap. The reality gap emerges from the divergence between the behavior of an agent in simulation and its behavior when deployed in the real world. However, precisely measuring this discrepancy is nontrivial. Since the true dynamics and reward function of the real environment are often unknown or only partially observable, directly comparing the simulated and real MDPs transition functions is infeasible. Moreover, the policies performance differences observed between the two domains can result from a combination of factors, including modeling inaccuracies, sensor noise, unmodeled delays or actuation errors. They can also come from the intrinsic architectures of the designed policies, including overfitting on the simulation. This makes it difficult to attribute the gap to specific sources or to isolate its magnitude. In practice, metrics for the reality gap must capture both functional differences in dynamics and the resulting impact on policy behavior.

1.2.2 Real-to-Sim

Even when a measure of the reality gap of some sort is available or estimated, the following challenge lies in leveraging it to perform real-to-sim transfer in a meaningful way. The core difficulty stems from transmitting this discrepancy into practical corrections of the simulator. Since the gap may come from complex differences in system dynamics or sensory observations, it is not straightforward to decide how the simulator should be adjusted to reduce those differences. Additionally, the relationship between simulator parameters and the induced gap is hard to formulate, making it difficult to establish a reliable correspondence. Without a clear mechanism to relate observed discrepancies to specific components of the simulation, attempts to improve the simulator may cause new mismatch in dynamics or the overfitting of its behavior to the observed real-world data.

1.2.3 Gathering data

While trying to estimate the reality gap to improve the simulator, another problem element that arises is the process of gathering data from real systems. Unlike simulation, where the desired data is computationally generated hence rapidly, without cost and accurately observable, real-world data collection is constrained by physical limitations, safety considerations, and sensors accuracy. Robots are subject to wear and mechanical fatigue, and repeated experiments can introduce long-term degradation or failure. Moreover, real environments can be subject to uncontrollable disturbances, making consistent data acquisition challenging. The desired data coming also from sensors that are not perfectly accurate, need a lot of processing. Sometime, the sensor for the direct data collection is not available, introducing the problem of estimating and inferring from other, measurable data. In addition, collecting sufficiently diverse and representative trajectories requires careful experimental design and execution. These constraints impose strict limits on the quantity and quality of data that can be obtained, which directly impacts the fidelity of any downstream modeling or adaptation processes.

1.3 Research objective

The main research objective of this master's degree was to to address the mentioned challenges posed by the reality gap in a RL paradigm, with a particular emphasis on finding solutions for the Reality to Simulation (Real-to-Sim) problem. This objective can be layered in the two following subsidiary objectives.

1.3.1 Real-world data collection

The first objective of this work that can be seen as a preliminary to the next one is to develop an approach to gathering real-world interaction data from physical robotic platforms. As we saw, unlike in simulation, data collection in the real world is not straightforward as it is facing various challenges. Therefore, a key component of the project is to study ways of collecting data by taking into account the robot constraints, the sensors available and to adapt this process to the purpose of the gathered dataset. Once collected, this data will be used to evaluate the extent of the reality gap by comparing observed real-world transitions and outcomes to those generated by the simulator. The research will explore quantitative metrics that capture state-transition discrepancies, action effects, and divergence in policy execution. To successfully achieve this objective, the following items are required:

- Design the process of selection of the real-world data subsets that will be collected.

- Design the process for collecting the chosen subsets of real-world data.
- Design the post-process methods to apply on the collected data.
- Execute the previous items on a real-world application.

1.3.2 Real-to-Sim process

The second and main objective of this research is to leverage the data obtained from this evaluation to systematically improve the simulator with the goal of producing more robust and transferable reinforcement learning policies. Rather than pursuing ad hoc tuning or manual calibration, the proposed approach will focus on using the measured discrepancies as a guide for modifying the simulation dynamics in a data-driven manner. This may involve adjusting transition functions, noise models, or other environment parameters to reduce the identified mismatch. By refining the simulator based on real-world observations, the aim is to create a training environment that better approximates reality while maintaining the efficiency and of simulated learning. RL policies trained in this enhanced simulator are expected to demonstrate improved generalization and reliability when deployed on the real robots. Ultimately, this research seeks to contribute a methodology for real-to-sim transfer, bridging the gap between theoretical advances in reinforcement learning and their practical deployment in real-world robotic systems. To successfully achieve this objective, the following items are required:

- Design an estimating model of the discrepancies between the real world and the simulation.
- Evaluate the improvement of the enhanced simulator from a reality gap perspective.
- Evaluate the improvement of the RL policies performances in the real world.
- Execute and evaluate the previous items on a real-world application.

1.4 Thesis outline

The rest of this thesis is structured as follows. Chapter 2 will be dedicated to a detailed review of existing methods and scientific advances in the literature. The following topics will be studied: RL on real-world applications, simulator enhancement, sim-to-real transfer, real-to-sim processes, Gaussian processes and specific experimental setups. Chapter 3 presents the research approach and thesis organization. Chapter 4 consists of the article that presents the method proposed, a framework for a Real-to-Sim process used for simulator enhancement. Chapter 5, finally, conclude our work by summarizing the ideas of the method developed, their strengths and limitations, and presents our recommendations for future works.

1.5 Scientific contribution

The scientific contribution of this thesis lies in the article presented in chapter 4. This article was submitted to the journal *IEEE Robotic and Automation Letters* and is currently under the review process. The literature review, conception of the paper, analysis and writing were done entirely by myself and the co-authors contributed for the real-world experiment setup and provided feedback on the writing.

CHAPTER 2 LITERATURE REVIEW

This chapter present an overview of the existing ideas, methods, and metrics that were relevant during the development of the work presented here or present similar concepts that we want to highlight. The covered concepts are the RL applications to real-world applications and their solutions to solve the reality gap problem. A specific section is dedicated to Gaussian processes and their application for building world-models.

2.1 Reinforcement Learning for Robotic Control and the Sim-to-Real Challenge

The Reinforcement Learning paradigm we place ourself in was presented in the book by Sutton and Barto [1]. The interest has skyrocketed since the demonstration of its performance on the well-known Atari 2600 benchmark [3]. More recent studies have demonstrated the potential of RL in continuous control using algorithms like Deep Deterministic Policy Gradient (DDPG) [4], Soft Actor-Critic (SAC) [5], and Proximal Policy Optimization (PPO) [6].

As RL offers a data-driven, model-free framework for learning policies directly from experience, it makes it especially attractive for tasks that are difficult to model explicitly such as robotics control, due to its sequential and dynamic nature. Applications include locomotion [7], manipulation [8], autonomous navigation, and aerial control. In these contexts, RL has enabled robots to learn to perform tasks such as grasping, walking, or flying, often surpassing human-designed controllers in adaptability and generality. An infamous example is the work presented in [9] where an RL policy is trained to control a race drone and it shows world champions race level, beating professional human pilots.

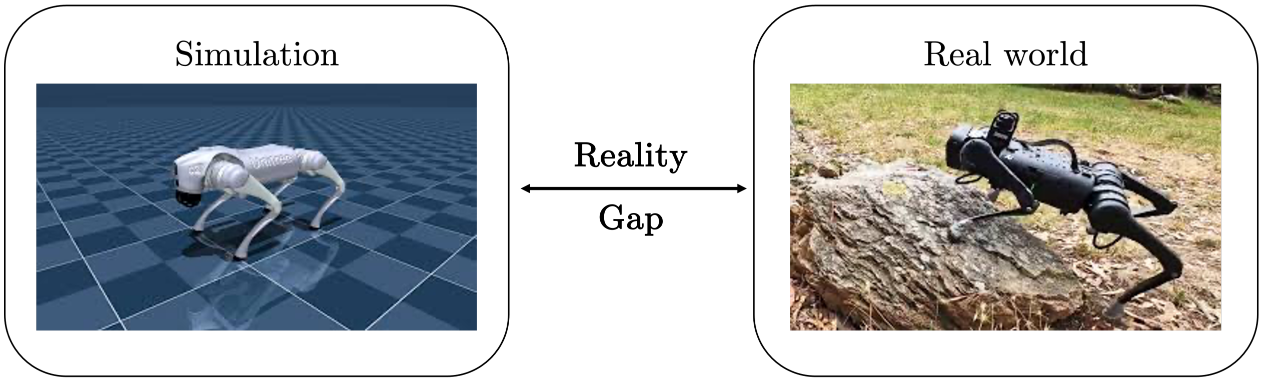


Figure 2.1 Reality Gap

However, transferring policies trained in simulation to physical robots remains a major bottleneck. This problem, known as the *sim-to-real* transfer problem, arises from the mismatch between the simulated environment and the physical world. leading to degraded performance when deploying the policies on hardware [10]. Recent literature has sought to address this challenge through different approaches that can be decomposed in two categories.

2.1.1 Sim-to-Real

The first categories of methods explore the problem from a direct approach, trying to setup the algorithms and simulator such as the transfer successfully happens. Several methods existing in the literature that aim to optimize Sim-to-Real are listed and compared in [11], [12].

In particular, domain randomization [13–15] consists of randomizing simulation parameters. The idea is that introducing stochasticity in the simulator will simulate the difference the policy will have to handle when deployed in the real-world, which is supposed to produce policies more robust to the real-world deployment. [16] has shown that for some application, the realistic part of the data is not essential for a Deep Learning model to learn as soon as the distribution of the task is correctly represented. The range of the parameters to randomize can be determined by the user before the training as in [17] or dynamically like in [18] where the subsets of parameters that presents creates harder environment are more often selected as the correspond to more "useful" environments.

Representation learning is another trail that has been explored in [19] by training a invariant feature model that is supposed to understand the common features between two environments to enable the transfer between them. Policy ensembles - training multiple policies with varying conditions and merging them - is also a possibility as presented in [20].

2.1.2 Real-to-Sim

On the other hand, Real-to-Sim aims to improve in some way the simulator to make it more realistic. A well-known example of this is the very generic idea of system identification [21] where the goal is to tune the simulator parameters to match real-world data. The authors of [22] find the value of those parameters through an iterative Newton-Euler algorithm. They can also be estimated with deep neural networks [23, 24]. Some methods mix the ideas of identifying system parameters and domain randomization such as in [25] where real-world data is used to guide the selection of the parameters range so that they create environments that show realistic behavior while producing robust policies. However, those methods are constrained by the model form that is used in the first place since they are only modifying parameters.

To get rid of this constraint, research is also done in the direction of learning directly world dynamics instead of some parameters of a defined model as in [26, 27]. In this paradigm, the system dynamics can be radically modified, allowing to start from a simulator that is completely idealized. The model estimating the dynamics of the target system can also focus only on the residual physics. This allows the leverage of a simulator that was tuned by only applying a correction to it.

To model the correction, we can have another hand-tuned model that was created by analyzing the real system such as in [28] where, after a careful system identification of the simulator, the authors observed that the policy was still unable to be transferred on the robot. They realized that it was due to the idealization of the motors providing a linear torque-relation. By experiments on the concerned motors, they developed a non-linear correction term that they input directly during the simulator step to generate realistic behavior, allowing the newly trained policy to be deployed on the real robot. While allowing to have different dynamics (changing linearity for instance) as opposed to tuning the parameters of the system, this method still suffer from the same problem of being highly task-dependent and needing a heavy engineering knowledge of the dynamics that are trying to be modeled.

To solve this, the work in [29] propose to use an RL policy to estimate the residual forces that need to be applied on the system. They demonstrate their method on balloon robots, showing good results even in the case of highly non-linear dynamics.

2.2 Gaussian Processes

Gaussian Processes (GPs) have been proved to be a powerful non-parametric tool for regression. They can be used to model complex dynamical systems, particularly in scenarios where prior knowledge is limited and data is scarce. Initially presented in [30], it has been widely adopted due to the work in [31]. Their efficiency was proven and compared to neural network's in [32] which motivated the deeper look into using it for estimating the dynamic systems.

2.2.1 Theory

The mathematical formulation that will be presented here is detailed in the book [33]. Gaussian Processes (GPs) are strong non-parametric models that are widely used for regression [31]. Unlike parametric models, GPs define a prior directly over functions, allowing modeling without the need of assumptions on the estimated function form which makes it ideal for estimating dynamics that are completely unknown.

Definition 1. Gaussian Process [34]

- A random variable $\mathbf{X} \in \mathbb{R}^n$ follows a **Multivariate Gaussian Distribution** in n dimensions of mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ if its density function is given by
$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp(-\frac{(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}{2})$$
We note $\mathbf{X} \hookrightarrow \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- A **Stochastic Process** is a collection of random variables $\{X_t, t \in T\}$ indexed by T and taking their values in a state space S.
- A Stochastic Process $\{X_t, t \in T\}$ is a **Gaussian Process** if, and only if, for every finite subset $t_1, ..., t_k$ of T, the random variable $\mathbf{X}_{t_1,...,t_k} = \{X_{t_1}, ..., X_{t_k}\}$ follows a Multivariate Gaussian Distribution.

Given those definitions, a Gaussian Process can be defined by its mean function:

$$\begin{aligned} \mu : T &\to \mathbb{R} \\ t &\mapsto \mathbb{E}[X_t] \end{aligned}$$

and its covariance function:

$$\begin{aligned} k : T \times T &\to \mathbb{R} \\ t_1, t_2 &\mapsto \mathbb{E}[(X_{t_1} - \mu(t_1))(X_{t_2} - \mu(t_2))] \end{aligned}$$

In this work, our use of Gaussian process will only concern **real-valued random field process**, *i.e.* $T = \mathbb{R}^d$ and $S = \mathbb{R}$. Starting from here, we will switch to notation that fits more the specific types of spaces we are dealing with:

- Elements of the index set T will be denoted as $\boldsymbol{x}$ as they represents vector of $\mathbb{R}^d$, potentially with subscript or superscript depending on the need.
- The random variable associated to the index element $\boldsymbol{x_i}$ will be denoted as $Y_{\boldsymbol{x_i}}$
- Elements of the state space S (the values taken by the random variables) will be denoted by y_i, being scalars.

Now let $f : \mathbb{R}^d \to \mathbb{R}$ be an unknown function we aim to estimate based on observations. We can use the Gaussian Process random variables to represents the function outputs. The Gaussian process produces then an estimation of the function:

$$\hat{f} \sim \mathcal{GP}(\mu(\cdot), k(\cdot, \cdot)) \tag{2.1}$$

The mean function represents our prior knowledge on the function we are trying to estimate. In many case, we don't have this knowledge so we will assume a zero mean function. The kernel function, on the other hand, represents how the outputs are correlated according to the closeness of the inputs:

$$\mu(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] = 0 \tag{2.2}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] \tag{2.3}$$

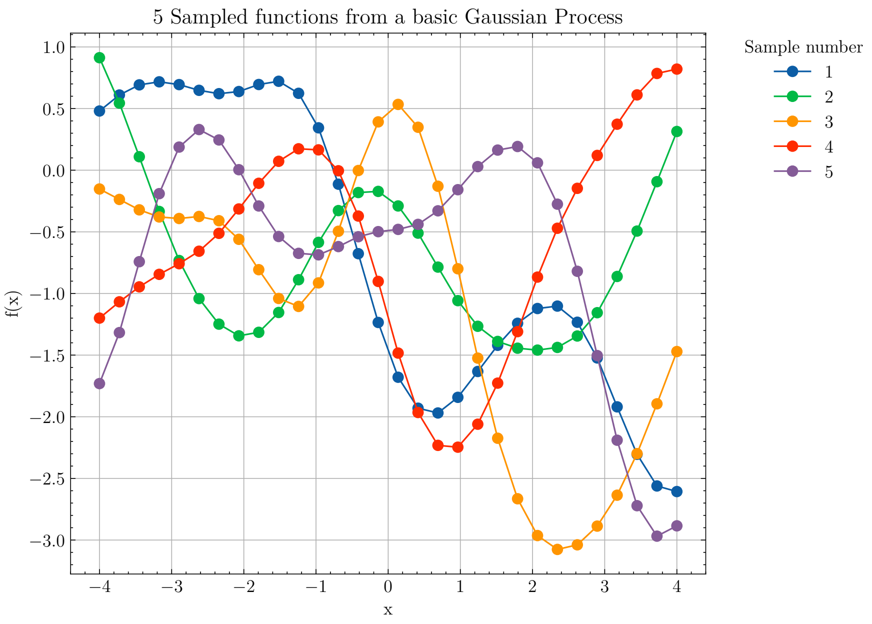


Figure 2.2 Sampling function from a Gaussian process

The GP will not give us a closed-form for a function estimate that we could be able to use out-of-the-box, instead, it will allow us to sample the estimated function values for any finite number of test points. Indeed, let us recall that, for any finite subset $x_1, ..., x_n$ of $\mathbb{R}^d$ the random vector $\boldsymbol{Y}_{t_1,...,t_n}$ follows a Multivariate Gaussian Distribution. Which means we can draw a sample for this random vector, using the mean and covariance function of the GP.

Such samples are shown in figure 1. We drawn 5 samples, each containing 20 test points $x_1 = 0, x_2 = 1, ...x_{20} = 19$. The mean prior is considered to be null: $\mu(\boldsymbol{x}) = 0, \forall \boldsymbol{x} \in \mathbb{R}^d$, and the kernel is the Radial Basis Function (RBF), the most common choice in Gaussian Process: $k(x_1, x_2) = \exp(-\frac{\|x_1 - x_2\|_2}{2\sigma^2})$. Each curve represents a sample of the random vector $\boldsymbol{Y}_{\mathbf{0,...,19}}$ and can be seen as a function realization of the Gaussian process.

Now suppose we observe a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times d}$ and $\mathbf{y} = [y_1, \ldots, y_n]^T \in \mathbb{R}^n$, where

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{2.4}$$

And we want to evaluate the GP estimation of the function at a test point $\mathbf{x}_*$, we can construct the joint distribution of the training outputs and the desired estimation as:

$$\begin{bmatrix} \mathbf{y} \\ f(\mathbf{x}_*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} K + \sigma^2 I & k_*^T \\ k_* & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix} \right) \tag{2.5}$$

Where $K \in \mathbb{R}^{n \times n}$ is the covariance matrix of the training set $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, and $k_* = [k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_n)]$ is the vector of covariances between the test point and the training inputs.

Leveraging the property of multivariate Gaussian distribution we can derive the conditional distribution of $f(\mathbf{x}_*)$ given the training points as a Gaussian distribution of mean and variance [31]:

$$\hat{y}(\mathbf{x}_*) = k_*^T (K + \sigma^2 I)^{-1} \mathbf{y} \tag{2.6}$$

$$\sigma_{\hat{y}}^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - k_*^T (K + \sigma^2 I)^{-1} k_* \tag{2.7}$$

This allows us to predict the function value at any new input with an associated measure of uncertainty.

For the choice of the kernel, the recommendations from [31] are to use a sum of an RBF and a linear kernel. This Radial Basis Function (or exponentiated quadratic), is the most used kernel:

$$k(\mathbf{x}, \mathbf{x}') = \alpha \, \mathbf{x}^\top \mathbf{x}' + \beta \, \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2} \right) \tag{2.8}$$

Where α and β are amplitude terms and σ is the lengthscale of the RBF kernel.

We show in the figure 2.3 the behavior of the RBF kernel for two particular lengthscales. And on figure 2.4 for each one a function sampled from a Gaussian Process with the corresponding kernel.

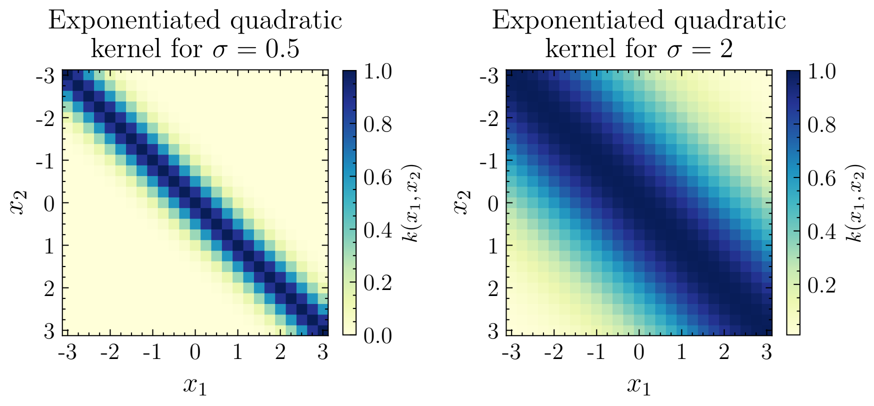


Figure 2.3 RBF kernels for different lenghtscales

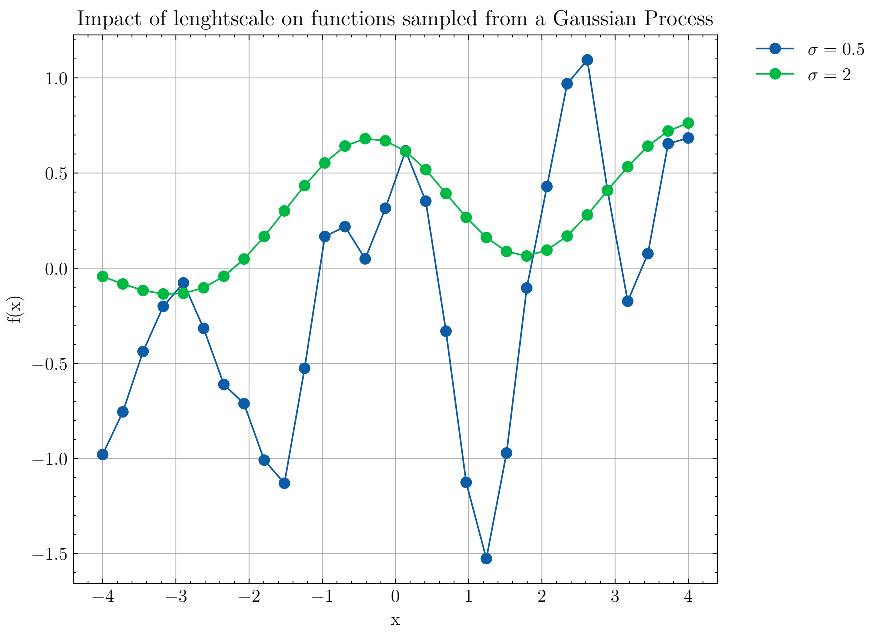


Figure 2.4 Impact of σ on the functions sampled from a Gaussian process

In the case of a multidimensional input ($\mathbf{x} \in \mathbb{R}^D$), we can generalize equation (2.8) by separating the lengthscale dimension-wise:

$$k(\mathbf{x}, \mathbf{x}') = \beta^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x'_d)^2}{\sigma_d^2}\right) + \sum_{d=1}^{D}\alpha_d^2 x_d x'_d \tag{2.9}$$

This process is known as Automatic Relevance Determination [35] and allow the kernel to select one lengthscale for each input dimension, enabling more precise function estimation.

But there are a tremendous amount of different kernel and choosing among them can be very complicated. Such a list of kernels, how they behave and the impact on the estimation produced can be found in the 4th chapter of the book [33] and in [36].

The lengthscales σ_d, amplitudes β, α_d and noise σ are hyperparameters of the GP that can be learned by maximizing the log-likelihood of the training data [31]:

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T(K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1}\mathbf{y} - \\ \frac{1}{2}\log\det(K_{\boldsymbol{\theta}} + \sigma^2 I) - \frac{n}{2}\log 2\pi\end{aligned} \tag{2.10}$$

Here, $K_{\boldsymbol{\theta}}$ denotes the covariance matrix constructed from the kernel function parameterized by $\boldsymbol{\theta}$. Maximizing this log-likelihood balances data fit with the model complexity, while accounting for uncertainty in the predictions. This optimization is can typically be done using gradient-based methods, as the log-likelihood is differentiable with respect to the hyperparameters.

2.2.2 GP in RL

In the context of robotics and reinforcement learning, a whole class of articles use GPs for value-function estimation as it is theoretically shown in [37]. Here, the GP does not estimate directly the system dynamics but the behavior of the interactions between the agent and the environment in terms of reward. In [38], the GP is used to estimate the next state to compute the policy update.

Let us now assume that the world dynamics can be formulated as a discrete time process: At a given time t, if the state of the studied system is denoted s_t and the control action we apply to it u_t, the state of the system at the next timestep $t + 1$ will be given by:

$$s_{t+1} = f(s_t, u_t)$$

Where f is the *transition function* of the process.

A GP can be used to estimate this transition function f as it is shown in [39–41]. [42] couple this transition function estimation to a model predictive control optimization [43] to learn a controller.

This prediction of the next state can be extended to a multi-step forecasting process. In [44], this is done by successively feeding the GP with the state we want to start from and then keep on predicting by providing the previous output as a new input until we reach the desired horizon.

However, those methods are trying to create a model that estimate the whole trajectory whereas a lot of work has already been done to produce effective and accurate simulator. To leverage this idea, we can use the GP to estimate the residual dynamic of this simulator. This was done initially in [45] where the authors estimate the residual dynamic of a blimp simulator according to real-world data. The work presented here demonstrate that their method finds applications that goes beyond what they initially presented.

2.3 Experiments

This section presents the works and literature that were used to build the experiments conducted to demonstrate the effectiveness of the method.

2.3.1 CartPole environment

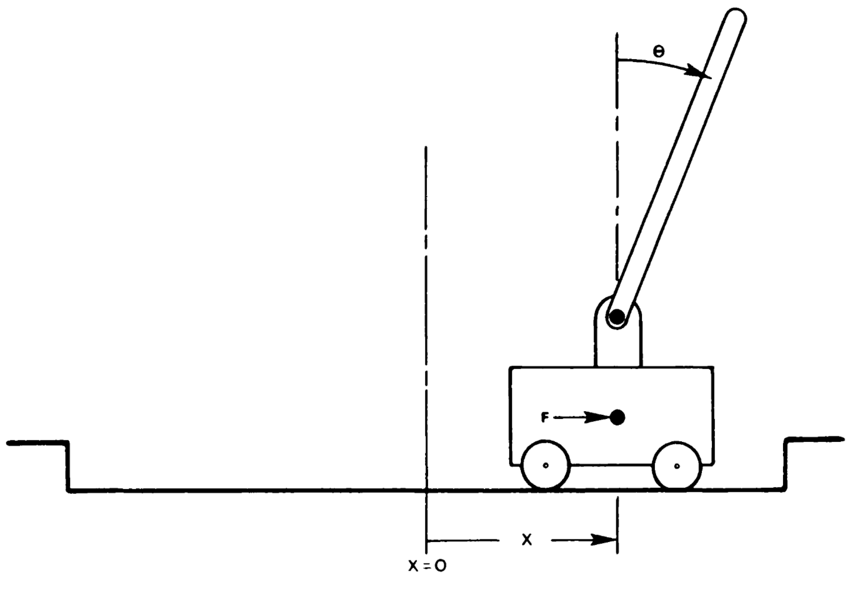


Figure 2.5 The CartPole system, as presented in [2]

RL has benefited from the unification of the environments API under the Open-AI gymnasium library [46]. In particular, the control environments such as the CartPole in provide a simple yet effective environment that can be used as a toy case or motivation experiment. This environment was firstly used in [2] to demonstrate RL agents performances. The task consists of maintaining the pole in a vertical position by providing left-right control actions to the cart. Extending the gymnasium's implementation by adding friction, the full physic equations ruling this system are detailed in [47].

2.3.2 Unicycle and Limo

Unicycle

The second and main experiments aims to transfer an RL policy from a unicycle environment to a Limo in ackermann drive mode.

The unicycle environment is the one that came from [48]. It consists of a basic kinematic simulator modelizing the behavior of an idealized unicycle agent controlled by linear and angular velocities inputs. The task is to reach a goal placed randomly in the arena.

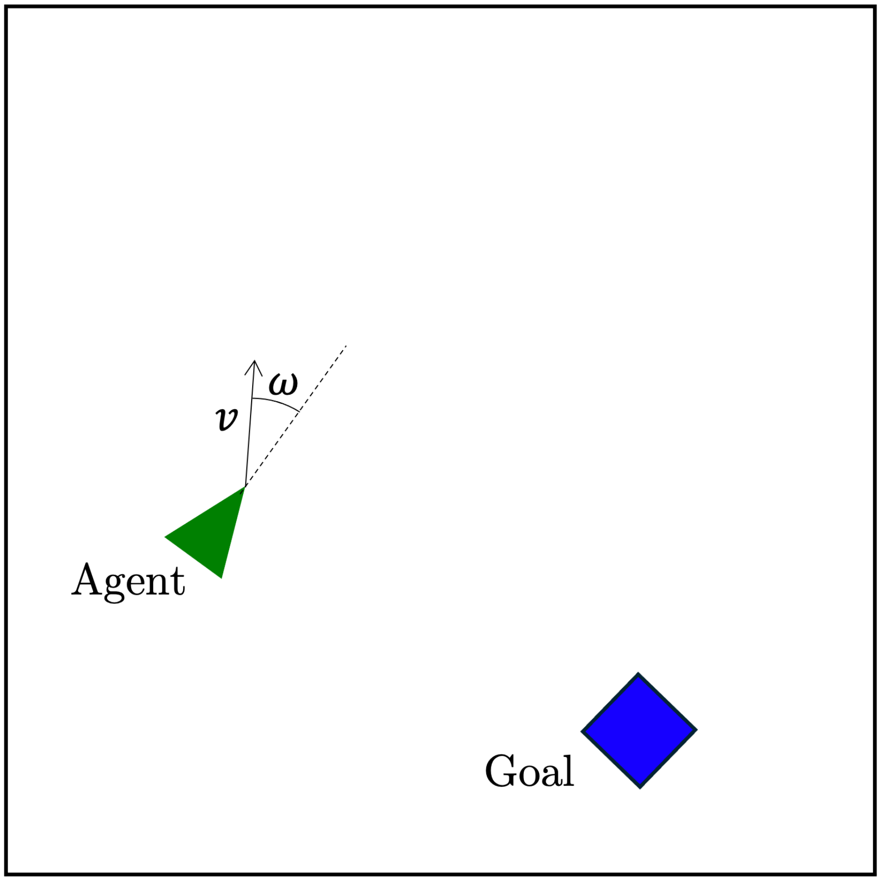


Figure 2.6 The Unicycle environment

Limo

The target environment for this sim-to-real task is a similar arena in which we place a Limo from the company Agile X [49], in ackermann drive mode [50]. This drive mode is the most familiar setup as it is the one that rule our car steering mode. The front wheels are allowed to rotate, imposing a steering angle, and the back wheels impulse a linear speed. The goal is virtual and is position is also determined randomly at the beginning of the episode.

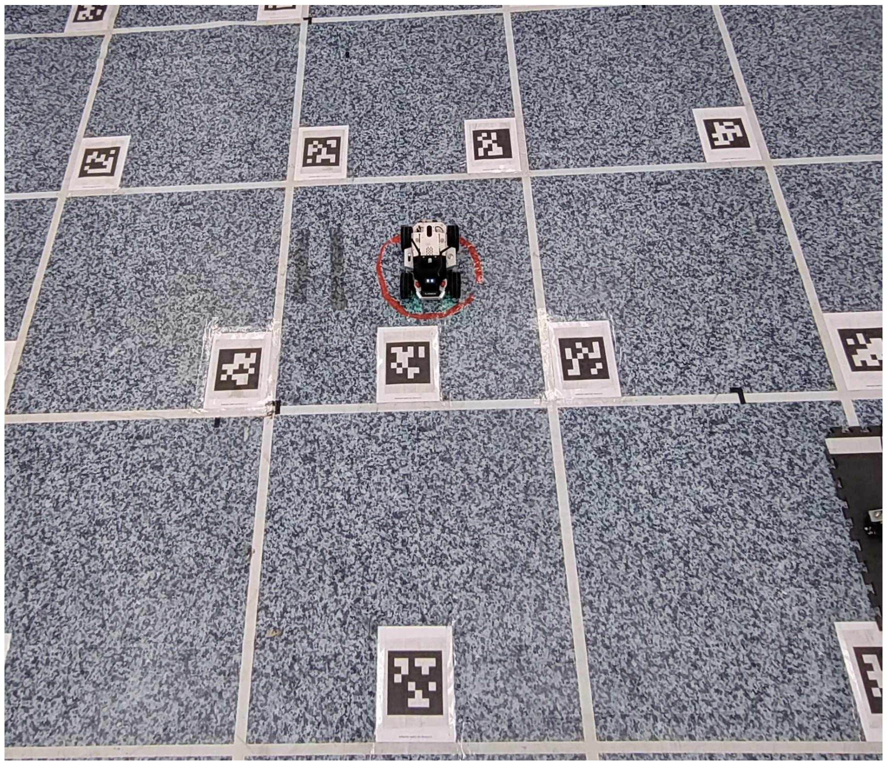


Figure 2.7 The Limo in its arena

The Limo control is operated through the Robot Operating System (ROS) [51] framework. ROS is a flexible and modular middleware widely used in robotics for integrating control, perception, and planning components. It provides standardized communication tools and libraries that unify the development of complex robotic systems.

CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION

This chapter will present the research approach used to meet the objectives detailed in section 1.3. We will first detail how it link with the article of chapter 5 and then present the document structure.

3.1 Research approach

The work that was conducted during the master's degree and presented in this thesis was a part of a bigger project, ENVIA, coming from a collaboration between Flying Whales Québec and MISTLab as it is detailed in section 1.1.4.. Isolating a subset of this project allowed to discover the research objectives that guided the work that lead to the writing of an article. This project's subset consist of focusing on finding a way to perform simulator enhancement in a real-to-sim fashion. This objective in mind, a method using Gaussian processes to estimate residual physics was found and demonstrated to allow not only to reduce the reality gap but also to produce RL policies robust to Sim-to-Real transfer. This method is designed to be used regardless of the form of the dynamics unmodeled by the simulator as we shown that it was efficient even in case of deep mismatches between the simulation and the real world. As the MISTLab expertise lies in robotic control, we made sure to test the method with an experiment on real robots.

3.2 Document Structure

The document's structure follows the one of a thesis by articles. Because I am first author for the article presented, this one will be directly included, as is, in the body of the thesis under its original article format. The document is structured as follows:

- Chapter 1 gives an introduction on the context of research, the key concepts that build it and the research objectives.
- Chapter 2 presents an overview of the literature relevant for the understanding of this thesis.
- Chapter 3 details the organization of the research and the document structure.
- Chapter 4 consists of the article presenting the method for simulator enhancement with Gaussian processes.

- Chapter 5 summarizes the thesis and provide discussions about the results and the future works.

CHAPTER 4 ARTICLE 1: CLOSING THE GAP: GAUSSIAN PROCESSES ENHANCED SIMULATION FOR REINFORCEMENT LEARNING POLICY TRANSFER

Preface: This paper was conducted by myself as a first author. The conception, literature review, experimental setup and result analysis were my own production. I received help and guidance on the writing from the secondary authors.
Full citation: Clement Garancini, Maeva Guerrier, Hassan Fouad, Giovanni Beltrame, "Closing the GaP: Gaussian Processes enhanced simulation for Reinforcement Learning policy transfer", *IEEE Robotics and Automation Letters*, under review.
Submission: June 18th, 2025

Abstract

The recent success of Reinforcement Learning (RL) has made it a promising candidate for real-world application such as robotic control. The inherent limitations of training in real life and the will to leverage the computational resources at our disposal paved the way for the use of simulation to train policies that are then deployed to the real robots (*sim-to-real*). However, despite increasingly powerful simulators and sim-to-real techniques, transferability of RL policies toward systems that show highly complex dynamics is still an open problem. In this work, we demonstrate that a method using Gaussian Process (GP) to accurately estimate residuals between a simulator and the real world can close the sim-to-real gap. We show through simulation and hardware experiments that the policies trained using this GP-enhanced simulator successfully solve the task in the target environment even when there is a large mismatch between simulation and real world dynamics.

Keywords

Reinforcement Learning, Gaussian Process, Real-to-Sim, Reality Gap

4.1 Introduction

Simulators are widely used to solve the generic problem of training RL policies to control real-life robots due to the many constraints that real-life training presents such as safety, time, space, energy and robot costs [9, 52, 53]. However, the discrepancies between the simulator and the real world, or so-called *reality gap*, hinders the transferability of policies trained in simulation to the real world.

Simulation and reality, are connected via two links: the *sim-to-real* and the *real-to-sim* transfers. Sim-to-real is concerned with real-life deployment of a policy trained in simulation, while real-to-sim aims to use data from real world to improve the simulation in which a policy is trained. [11] and [12] list and compare several sim-to-real methods existing in the literature. In particular, domain randomization [13,14,18] consists of randomizing simulation parameters, which leads to more robust policies showing good performance after sim-to-real transfer.

On the other hand, real-to-sim leverages several approaches like system identification [21, 54] or learning methods that model the world dynamics [26]. Some methods merge the sim-to-real and real-to-sim such as [22,25], making a sim-to-real-to-sim framework. However, instead of learning a world model from scratch one could directly leverage the simulator dynamics to build a residual model estimating the difference between the existing simulator and the real world [29, 55].

A practical concern in the sim-to-real process is the selection of the best simulator for the task. Many popular simulators such as MujoCo [56], Isaac-gym [57] or Gazebo [58] are cumbersome to setup, require customization to fit the real-world and need a tremendous amount of computational resources to run. Also because of their generic design to fit a wide range of tasks, they may not be adapted to the desired task. The other option is to start with a simple simulator; that is easy to use and computationally fast; irrespective of the initial sim-to-real gap and enhancing it to bring it closer to the real-world. Moreover, some works demonstrated the potential of using such simple simulators to quickly train policies for sim-to-real transfer [59].

Gaussian Process (GP) is a powerful non-parametric and sample-efficient regression tool that was widely adopted for estimating function after the work of [31, 32]. [39–41] showed that GPs could successfully model the dynamics of a system and form a simulator close to the real world. Combined with the idea of estimating the residual dynamics, [60] show that one can use GP to estimate the residual dynamics of the simulator and then use it to enhance this very same simulator.

We present in this paper a sim-to-real-to-sim framework where a first sim-to-real transfer is used to collect data from the real world. A GP is then trained to estimate the residual errors between the forward dynamics predictions of the simulator and the real world for the real data previously collected. To perform the real-to-sim, the GP trained is used to enhance the simulator and train a new RL policy that will finally be deployed in a final sim-to-real

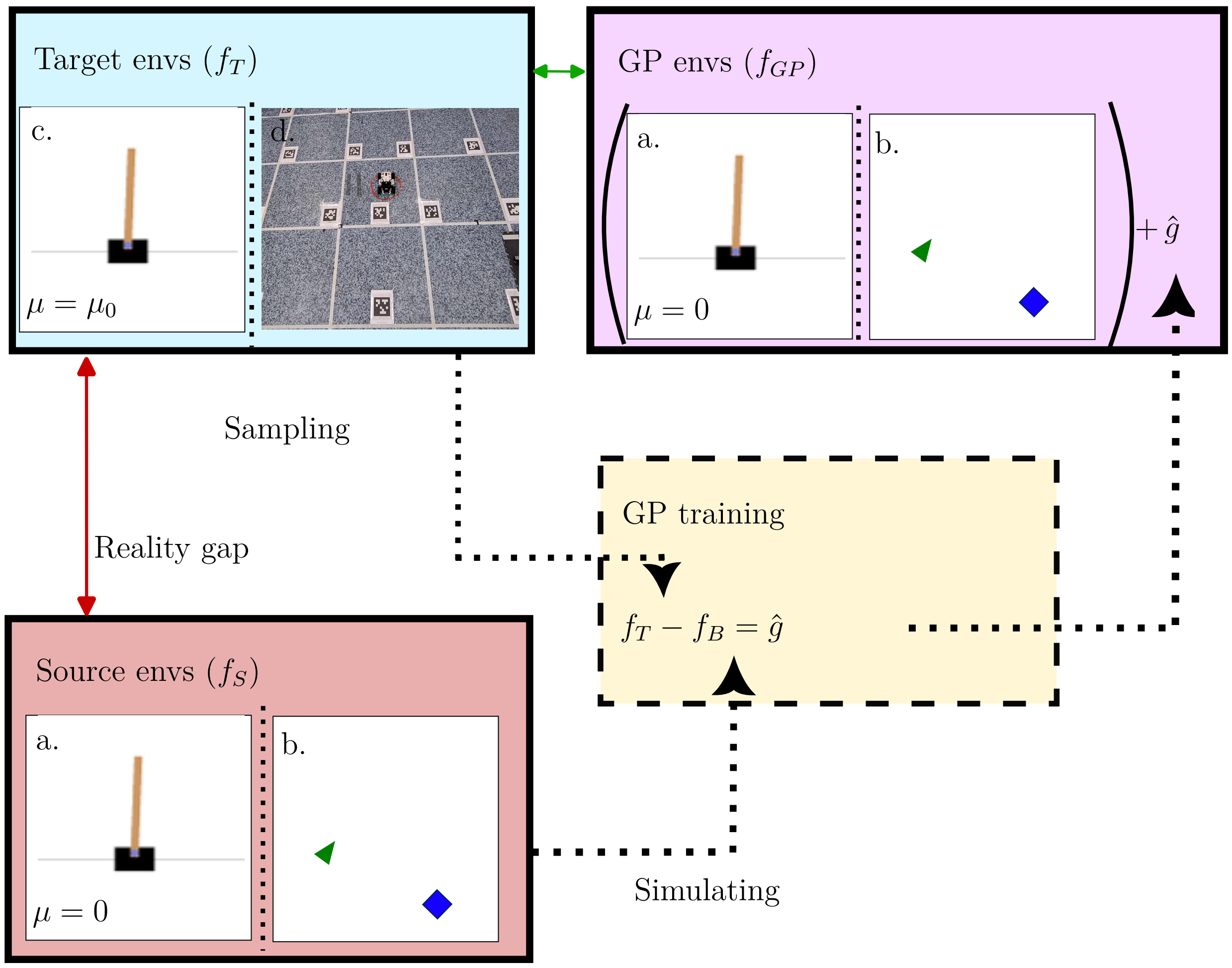


Figure 4.1 Framework scheme for simulator enhancement with Gaussian Process. a. CartPole (Source). b. Unicycle (Source). c. CartPole (Target). d. Limo (Target)

transfer. Our contribution can be summarized in the following points:

- While our RL-GP framework builds on the work of [60], their work is limited. We are addressing those limitations here. Before applying the GP enhancement, they developed a sophisticated simulator, tuned by hand for their specific task. In our work on the contrary, we show that the GP is able to close the gap even when the initial simulator is very simple and presents large dynamic mismatches. Moreover, they did not evaluate the generalization of the method over various initial conditions task did not show any randomness in the initial conditions and the goal to reach, where we showed that the GP was able to work in all the environment conditions.

- We show that having the smallest reality gap do not yields better policy performance in general and that GP could leverage the uncertainty in their prediction to produce more stochastic environments which make better policy performance.

- We confirmed that the method presented can be applied on real-world RL tasks in credible settings with an experiment using a 4-wheeled AgileX's Limo in a goal-reaching task. Not only does the GP error correction generates realistic trajectories, but it allowed the training of a policy that successfully solve the task in the real world.

4.2 Preliminaries

4.2.1 Discrete time process dynamics

In this work, we will assume that the world dynamics can be formulated as a discrete time process. At a given time t, if the state of the studied system is denoted s_t and the control action we apply to it u_t, the state of the system at the next timestep $t+1$ will be given by:

$$s_{t+1} = f(s_t, u_t)$$

Where f is the *transition function* of the process.

A simulator can be seen like such process, and in the scope of this work, we will denote the transition function of the Source environment (resp. target environment) by f_{base} (resp. f_{target}). In the case of sim-to-real problem, where the Source environment is the simulation and the target environment is the real world, the reality gap can be formulated as $e = f_{true} - f_{base}$.

4.2.2 Reinforcement Learning

We model the RL problem as an infinite-horizon discrete-time Markov Decision Process (MDP), denoted $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ [1]. At each time step $t \in \mathbb{R}^+$, the agent observes a state $s_t \in \mathcal{S}$ and selects an action a_t from a discrete action space $\mathcal{A} = \{0, 1, \ldots, K\}$ according to a stochastic policy $\pi_\theta(\cdot|s_t)$, parameterized by θ. The environment then transitions to the next state s_{t+1} via the unknown dynamics $\mathcal{P}(s_{t+1}|s_t, a_t)$, and the agent receives a reward $r_t = \mathcal{R}(s_t, a_t)$. The agent's objective is to learn the optimal policy parameters θ^* that maximize the expected cumulative discounted reward:

$$\pi^* = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \begin{array}{l} a_t \sim \pi_\theta(\cdot|s_t), \\ s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t) \end{array} \right] \tag{4.1}$$

Where $\gamma \in [0, 1[$ is the discount factor, giving more importance to immediate rewards.

4.2.3 Gaussian Process

Gaussian Processes (GPs) are strong non-parametric models that are widely used for regression [31]. Unlike parametric models, GPs define a prior directly over functions, allowing modeling without the need of assumptions on the estimated function form which makes it ideal for estimating dynamics that are completely unknown.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be an unknown function we aim to estimate based on observations. A Gaussian Process is a random process in which any finite subsets of the random variables, representing the function outputs, follow a multivariate Gaussian distribution. Formally, all the information of a GP is encapsulated in a mean function $m : \mathbb{R}^d \to \mathbb{R}$ and a covariance function (or kernel) $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.

$$\hat{f} \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot)) \tag{4.2}$$

Because the mean function represents our prior knowledge on the function we are trying to estimate, we will assume a zero mean function. The kernel function represents how the outputs are correlated according to the closeness of the inputs:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] = 0 \tag{4.3}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] \tag{4.4}$$

Now suppose we observe a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times d}$ and $\mathbf{y} = [y_1, \ldots, y_n]^T \in \mathbb{R}^n$, where

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{4.5}$$

And we want to evaluate the GP estimation of the function at a test point $\mathbf{x}_*$, we can construct the joint distribution of the training outputs and the desired estimation as:

$$\begin{bmatrix} \mathbf{y} \\ f(\mathbf{x}_*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} K + \sigma^2 I & k_*^T \\ k_* & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix} \right) \tag{4.6}$$

Where $K \in \mathbb{R}^{n \times n}$ is the covariance matrix of the training set $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, and $k_* = [k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_n)]$ is the vector of covariances between the test point and the training inputs.

Leveraging the property of multivariate Gaussian distribution we can derive the conditional distribution of $f(\mathbf{x}_*)$ given the training points as a Gaussian distribution of mean and variance:

$$\hat{y}(\mathbf{x}_*) = k_*^T (K + \sigma^2 I)^{-1} \mathbf{y} \tag{4.7}$$

$$\sigma_{\hat{y}}^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - k_*^T (K + \sigma^2 I)^{-1} k_* \tag{4.8}$$

This allows us to predict the function value at any new input with an associated measure of uncertainty.

Following the recommendations from [31], we composed our kernel as a sum of an Radial Basis Function kernel and a linear kernel:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{\ell_d^2}\right) + \sum_{d=1}^{D} \sigma_{l,d}^2\, x_d x'_d + \sigma_b^2 \tag{4.9}$$

The lengthscales l_d, amplitudes $\sigma_f, \sigma_{l,d}, \sigma_b$ and noise σ are hyperparameters of the GP that can be learned by maximizing the log-likelihood of the training data:

$$\begin{aligned} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^T (K_{\boldsymbol{\theta}} + \sigma^2 I)^{-1} \mathbf{y} - \\ \frac{1}{2} \log \det(K_{\boldsymbol{\theta}} + \sigma^2 I) - \frac{n}{2} \log 2\pi \end{aligned} \tag{4.10}$$

4.3 Methodology

4.3.1 Workflow

Our method starts by training an RL policy on the Source environment in order to create a baseline that will be used to build the rest of our workflow. This policy will be optimized for the idealized environment that is denoted by S. We can then perform a first transfer into the target environment T in order to collect roll-out trajectories and create a dataset $D = \{(s_t, a_t), s_{t+1}\}$. We can then apply to each of this dataset entries the transition function of the Source environment to generate $f_S(s_t, a_t)$, which represents the state that our Source environment would have outputted if we had chosen the action a_t while in the state s_t. This allows us to compute the prediction error of the Source environment, or residual:

$$e_t = s_{t+1} - f_S(s_t, a_t)$$

This compose a dataset that was created by coupling the inputs of the Source environment and the residuals: $D_r = \{(s_t, a_t), e_t\}$. Using this training dataset D_r, we can now train a Gaussian process by gradient-descent optimization on the GP hyperparameters with early-stopping based on the log-likelihood improvement.

This trained Gaussian process is then used to improve the Source environment transition function by creating a new transition function:

$$f_{GP} = f_B + g$$

The intuition here is that, at each timestep, the Gaussian process will evaluate the error of the Source environment prediction and apply the correction so that the resulting forward dynamics prediction match the target environment dynamics.

The f_{GP} thus created is then used in a new RL environment to train a policy that will learn to solve the task with constraints that are closer to the ones it will encounter in the target environment. The new environment, relying on the Source environment and enhanced by the GP error estimation, will be denoted in the rest of this work as the *GP environment*.

Once this new policy is trained, we can deploy it in the target environment to evaluate its performance.

4.3.2 Metrics

Evaluating the performance of our method is not straightforward as the task we are trying to solve, the sim-to-real transfer of an RL policy, was not directly optimized in our GP training process. Indeed, the GP was trained to estimate the error between a forward dynamic model prediction and some data gathered in a target environment and has, in essence, nothing to do with RL training. We can therefore break down our evaluation process in a 3-layered fashion:

Evaluating the GP estimation

The atomic step of the evaluation, and the most straightforward, is the evaluation of the direct prediction of the Gaussian process. This can be done easily by keeping some of the trajectories we gathered during the dataset collection as a test set D_{test} as it is done traditionally in any regression task. We can then compute the root mean squared error (RMSE) on D_{test}. This will help us make sure the that the GP has not overfit the training data.

Evaluating the reality gap

On an intermediary level rise the question of the gap that exist between the Source and the target environment. Since our goal is to bridge it by creating a GP environment we need a way to measure how close those environment are. To do so, we can, after collecting a trajectory in the target environment, initialize both the Source and GP environment with the initial state of the trajectory and feed them with the same actions that were taken, we can then measure the difference in prediction between the different environment.

Evaluating the policy performance

The last stage of our evaluation process is the evaluation of the trained policies' performances on the target environment. This evaluation is the most valuable as it directly corresponds to the fundamental motivations of this work. Indeed, whereas the two previous metrics will allow us to measure the soundness of our method, this one will evaluate the effectiveness of it.

4.4 Experimental setup

4.4.1 The CartPole, a toy case

The first experiment we conducted is a toy case providing motivations for the pursuit of this work. In this case, both the Source and Target environments are done in simulation. They rely on the gymnasium's [46] CartPole environment, a classic control RL environment. Whereas the environment that is in the gymnasium library does does not implement any form of friction, the complete equations can be found at [47]. We can then use the equations of the CartPole with friction as a target environment to simulate a real-world environment presenting that the Source environment (CartPole without friction) lacks to implement.

Environment formulation

The observation space of the CartPole is 4 dimensional $S = \mathbb{R}^4$, containing states $s_t = (x_t, \dot{x}_t, \theta_t, \dot{\theta}_t)$, where x_t and $\dot{x}_t$ are linear position and velocity of the Cart along the axis and θ_t and $\dot{\theta}_t$ are the angle and angular velocity between the pole and the vertical axis. The action a_t is taken in a discrete action space $A = \{0, 1\}$ pushing the cart towards the left or the right. At each timestep, the agent receives a reward $r = +1$. If the pole angle goes beyond a threshold θ_{safe} (*i.e.* if $|\theta| > \theta_{safe}$, the episode is truncated. The full equations governing the evolution of the CartPole's state are [47]:

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left(\delta + \mu g\,\mathrm{sgn}(\dot{x})\right)}{l\left(\frac{4}{3} - \frac{m_p\cos^2\theta}{m_c+m_p}\left[1 - \mu\,\sin\theta\,\mathrm{sgn}(\dot{x})\right]\right)}$$
$$\text{Where} \tag{4.11}$$
$$\delta = \frac{-F - m_p l\dot{\theta}^2\left[\sin\theta + \mu\,\mathrm{sgn}(\dot{x})\cos\theta\right]}{m_c + m_p}$$

$$\ddot{x} = \frac{F + m_p l\left(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta\right) - \mu N_c\,\mathrm{sgn}(N_c\dot{x})}{m_c + m_p} \tag{4.12}$$

Where N_c is the normal reaction of the ground on the cart:

$$N_c = (m_c + m_p)g - m_p l\left(\ddot{\theta}\sin\theta + \dot{\theta}^2\cos\theta\right) \tag{4.13}$$

Given the state of the system $s_t = (x_t, \dot{x}_t, \theta, \dot{\theta}_t)$ at a timestep t we can then use equations (4.11), (4.12) and (4.13) to compute the accelerations $\ddot{\theta}_{t+1}$ and $\ddot{x}_{t+1}$ and perform explicit Euler integration to get the next state s_{t+1} of the system. Because this integration step is similar in the Source and target environments' transition functions, we can reduce problem to the transition functions defined by $f_\mu(x_t, \dot{x}_t, \theta, \dot{\theta}_t) = (\ddot{x}_{t+1}, \ddot{\theta}_{t+1})$. We used the subscript μ to denote the difference between the function of the Source environment (f_{μ_B}) and the function of the target environment (f_{μ_T}).

Gaussian process enhancement

As we detailed in the section III.A we used the Source environment to train a policy π_B, to do so, we used stable baselines' [61] implementation of Proximal Policy Optimization (PPO) [6], a well-known RL algorithm that shows extremely robust performances. We trained on 10 parallels environment for a total of 300.000 steps.

Using this policy π_B we collected trajectories on the target environment over 20 episodes, and sub-sampled a dataset $D_T = \{s_i; f_T(s_i)\}_{i=1}^N$ where $N = 250$. We used this dataset to compute the GP training set $D_{GP} = \{s_i; f_T(s_i) - f_B(s_i)\}_{i=1}^N$. Because f_T and f_B are

Table 4.1 Cartpole environment parameters

Parameter	**Value**
Gravity (g)	9.8 N/kg
Cart mass (m_c)	1.0 kg
Pole mass (m_p)	0.1 kg
Pole length (l)	1.0 m
Force magnitude ($\|F\|$)	10 N
Source environment friction (μ_B)	0
Target environment friction (μ_T)	0.9
Time step (Δ_t)	0.02 s
Angle threshold (θ_{safe})	0.2 rad
Maximum steps	500
Total training steps (T)	3e5

vector functions of output dimensions 2 (linear and angular accelerations), we learned the hyperparameters of two GPs, g_x and g_θ, one for each of the acceleration component. We then built the GP environment upon the Source environment by creating the new transition function $f_{GP} : (s_t) \mapsto (f_B^x(s_t) + g_x(s_t),\ f_B^\theta(s_t) + g_\theta(s_t))$.

We then trained a policy π_{GP} on this newly created GP environment with the same setup as for π_B and deployed it on the target environment for evaluation.

4.4.2 The Limo, from Unicycle to Ackermann

Source Environment - Simulation

This task was build on the prior work of [48], a goal-reaching objective in a RL setting. The agent dynamics extend [48], implementing a unicycle agent that control freely both the linear and angular velocity (see Table 4.2). This is an RL episodic setting, where the episode ends if the agent reaches the goal or if the number of maximal time step is reached. Upon reaching the goal, the agent receives a scalar reward of +10. Throughout the episode, at each time step, the agent also receives a reward based on the variation of angular distance d^ϕ it has with the goal. Formally we have:

$$r_{orientation} = |d^\phi_{t-1}| - |d^\phi_t| \tag{4.14}$$

Where $d_t^{\phi} \in [-\pi, \pi]$. A positive reward meaning a decrease in the orientation distance, it provides an incentive for the agent to head towards the goal without penalizing the agent's initial pose (*e.g, initial poses where the agent is looking at the opposite direction of the goal*). The observation space is a 2D vector composed of the euclidean coordinates of the goal in the agent local frame of reference. Both the agent action (*linear and angular velocities*) and state space are continuous.

Table 4.2 Goal-reaching task Environment parameters

Parameter	**Value**
Arena size	1.5 m x 1.5 m
Goal radius	0.05 m
Agent radius	0.2 m
Linear velocity	[0.0, 0.2] m/s
Angular velocity	[-0.3, 0.3] rad/s
Simulation time step	0.1 (Δt)
Maximum steps	500
Total training steps (T)	4e6

Target Environment - Real world

For the Target environment, we use AgileX's Limo in Ackermann drive mode. This drive mode, essentially the one that we have on cars, allows the robot to control the front-wheels steering angle and the linear velocity, preventing the rover from rotating in place. The position coordinates, required for the observation space, are estimated by a motion capture system (OptiTrack Cameras). The episode is truncated if the agent goes outside of the arena. *Note that in simulation there are no truncation in this case.* The action are sent to the robot via the ROS1 interface of the robot through the \cmd_vel topic at a control rate of 10Hz.

GP Environment - Bridging the Sim2Real Gap

The GP environment builds upon the Source environment by adding the GP prediction to the angular and linear velocity. Two GPs are thus trained, one for each of the velocity component, taking as input the agent's proposed actions and predicting a correction (*see sect. III*) applied to the simulation linear and angular velocities dynamics formulations, which are as simple as the agent action commands. To create the training dataset for the

GPs, we performed a sweep over the environment action space $[0.0, 0.2] \times [-0.3, 0.3]$ of size 10x10. Each action was held for 30 control timesteps to account for the input delay of the robot. After the tracking of the position and orientation of the robot, and offline estimation of the linear and angular velocity was done by smoothing the signal with a 1-dimensional gaussian filtering of standard deviation $\sigma = 2$ provided by the SciPy python package and then integrating it with a finite-difference method.

4.5 Results

4.5.1 Cartpole

The trajectories plotted in figure 4.2 shows visually how the GP bridged the reality gap in the CartPole experiment. To generate this plot we took a random initial state s_0 and sequence of actions $(a_0, ..., a_\mathcal{T})$, set each of the environments (Source, GP and Target) initial state to s_0 and the sequence of actions sampled. As one can see, even if there is a small drift, the GP environment's trajectory is way closer to the Target's one than the Source trajectory.

For a more quantitative evaluation, table 4.4 shows the performances of the two GP models in the prediction of the next step acceleration (linear and angular) errors. The two metrics we chose are the Normalized Root Mean Squared Error (min-max) (**NRMSE**) and the $\mathbf{R}^2$ score. The models were evaluated by running 50 episodes on the target environment and comparing the GP error estimation and the actual error between the target and Source functions. As a GP output is formally a Gaussian distribution, we can either decide to take the mean of this posterior or sample a value from it to apply the correction in the GP environment. As expected, the mean prediction setting yields better results in terms of pure regression performance.

Table 4.3 Evaluation of one-step GP error prediction

Component	**Prediction**	**NRMSE (%)**	$\mathbf{R}^2$
Linear	Mean	**1.67 ± 1.06%**	**0.998 ± 0.003**
	Sample	4.24 ± 0.39%	0.989 ± 0.003
Angular	Mean	**1.64 ± 0.98%**	**0.998 ± 0.003**
	Sample	4.42 ± 0.66%	0.989 ± 0.004

However, when it comes to policy performance, table 4.3 shows us that the policy that was trained on the GP environment that uses the sampling prediction method has a better performance in the target environment. This highlight why we claim that GPs are a good

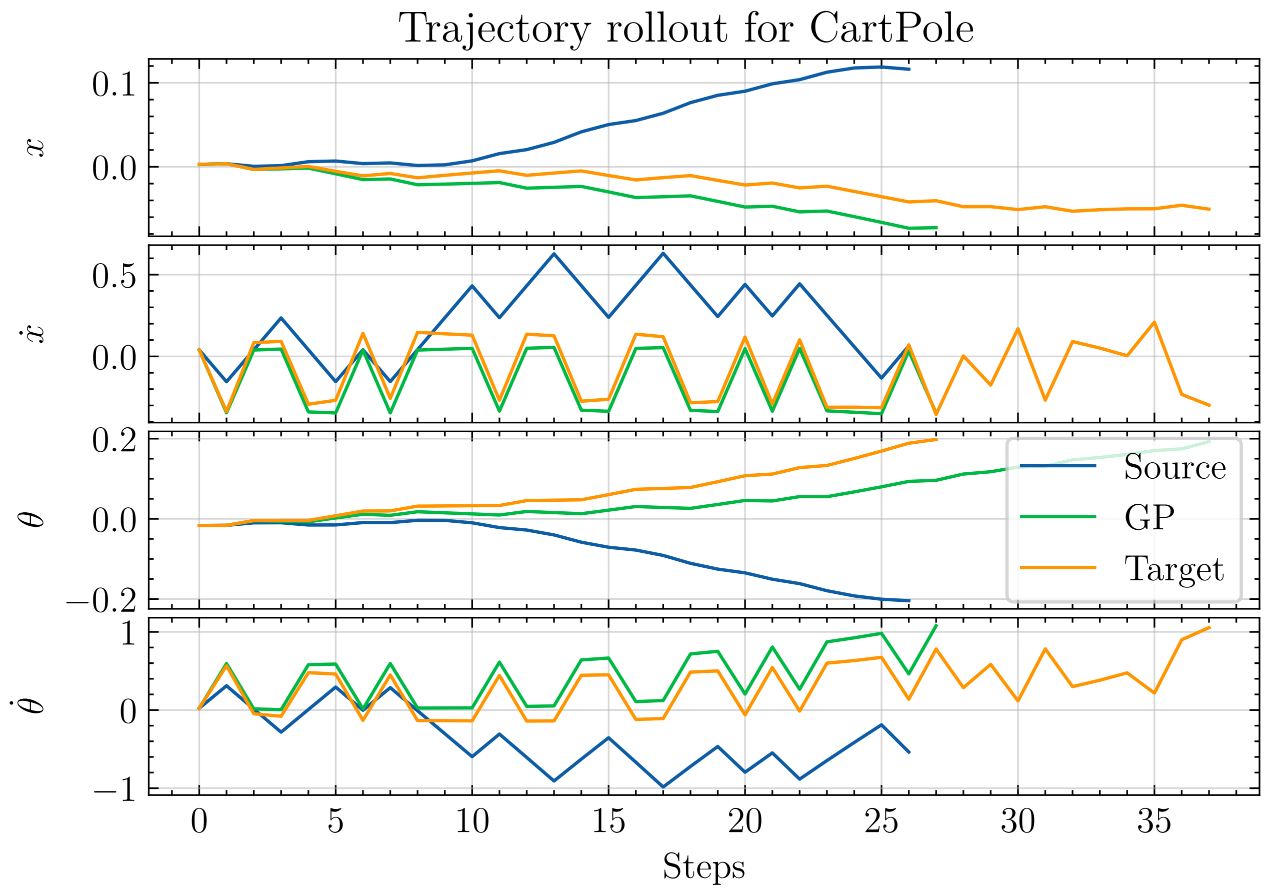


Figure 4.2 Rollout trajectory in different environments, the actions are randomly sampled from the action space.

fit for the sim-to-real problem. Indeed, when using it in the sample prediction setting, the variance of the GP prediction is propagated in the simulation process. This acts as a form of non-parametric, state-dependent domain randomization, producing a policy more robust to sim-to-real transfer.

We also did a very simple domain randomization environment in which we randomized the physic parameters of the Source environment uniformly in a $\pm 10\%$ range compared to the values defined in table 4.1. This policy, despite showing significantly better transferability than the Source policy, do not match the GP policies. This can be explained by the fact that we selected a high friction coefficient value (0.9), changing drastically the dynamics of the system. Even though we randomize the parameters, there is no set of parameters that can provide the same form of dynamics, making it hard for the policy to adapt to edge case where every action is crucial. As the Target environment is also run in simulation, we were able to train a "target" policy that we can consider as the optimal policy that we can reach while trying to solve this transfer problem.

Note: Due to the high level of friction we are applying to the cart, some of the initial conditions make the environment unsolvable because the pole has an angle too wide to being able to come back to an upward position, we excluded those episodes from the evaluation.

Table 4.4 Policy performance on target environment

Policy	**Episode return**
Source	144.62 ± 18.89
Domain Randomization	253.41 ± 54.26
GP-Mean	448.98 ± 134.79
GP-Sample	**497.86 ± 29.4**
Target	496.03 ± 38.89

4.5.2 Limo

The GPs estimation error is showed in table 4.5. The **NRMSE** is close between the mean and sample prediction settings because the variances of the trained GPs are shallow, causing the sampling process to output values close to the mean prediction.

Table 4.5 Evaluation of one-step GP error prediction

Component	**Prediction**	**NRMSE (%)**
Linear	Mean	**13.6 ± 3.9%**
	Sample	13.7 ± 3.8%
Angular	Mean	**15.3 ± 2.6%**
	Sample	15.4 ± 2.6%

To evaluate the proposed method in the Limo setup, we trained one RL policy π_S on the Source environment and another π_{GP} on the GP environment. Both training were done with the Soft-Actor-Critic (SAC) [5] implementation of stable baselines 3 [61] for 1.6 and 4 millions steps respectively.

We then deployed both policies on the real robot. Trajectories collected on the real robot are shown in fig. 4.3 in solid lines. In both case, we placed the agent near the center of the arena, facing different directions for each new trajectory. In all the trajectories collected, the goal was at the same position. *Note: During training, the initial pose of the agent and the position of the goal are randomly drawn at the beginning of the episode, this setup is for evaluation only.*. To visualize the impact of the GP enhancement of the simulator on the reality gap, we are showing the trajectories performed by the policies in their respective environment that corresponds to the same initial conditions (same agent pose and goal position). The trajectories' colors are representing the number of timesteps that were needed by the agent to reach the goal.

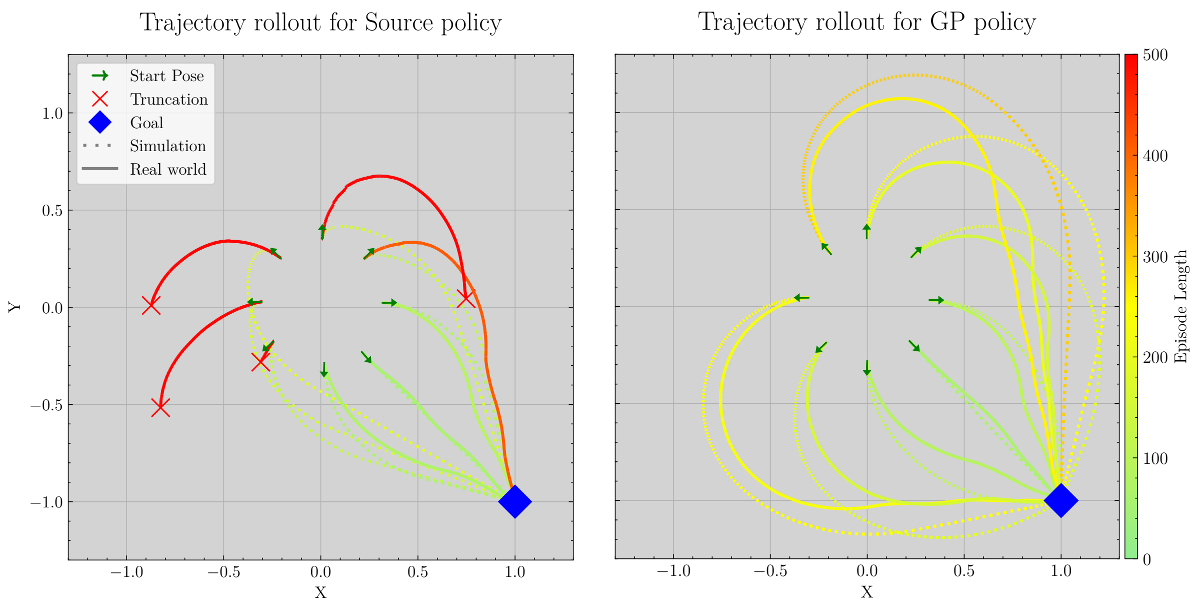
The left figure (4.3a) shows the trajectories that were followed by the Source policy π_S both in the Source environment (dotted lines) and in the real world (solid lines). In simulation, we can see that the policy rotates almost without moving and then goes towards the goal when it is facing it. In the real world however, the Ackermann mode does not allow the rotation in place. The policy tries to rotate with almost no velocity which forbid it to reach the goal within the 500 steps limit when the initial orientation of the agent is away from the goal (top left initial positions).

Now on the right figure, we show the GP policy π_{GP} trajectories in the GP environment (dotted lines) and in the real world (solid lines). Two points are conveyed with this plot. The first one is that the GP can indeed give a step-by-step correction that will produce more realistic trajectories. However, one can see that the simulated trajectories are making the agent taking wider turns which means that the GP is over-correcting the angular velocity. This comes from the way we created the estimations for angular and linear velocities as we waited for only 30 steps for the robot to reach its stabilized dynamic state, increasing the number of steps for each point in the action space sweep would give a better estimation but require more sampling time, we found that this sampling time was a good trade-off point to obtain good policy performance.

The second information is conveyed by the color of the real-world trajectories. We can not only see that the policy deployed can successfully solve the task for all the initial orientations, but it solve it for a near optimal number of steps, going almost at maximum velocity every time.

4.6 Discussion and Limitations

We showed with our two experiments, that a GP could be used to bridge the reality gap between a Source and a Target environment by estimating the error of the source's forward dynamic model and adding this correction term during the step-by-step simulation. The trajectories thus generated are closer to the reality bridging the gap from the simulator perspective. We also showed that using this *GP-enhanced* simulator for RL training permitted the successful transfer of the policies as they achieve near-optimal performance in the real-world. The specific formulation of the GP are a major reason for this success. Their non-



(a) (b)

Figure 4.3 Rollout trajectories in simulation and real world from different initial state. (a) The simulation is the Source environment and the policy used to rollout the trajectory is the Source policy. (b) The simulation is the GP environment and the policy is the GP policy.

parametric form make them ideal for to modeling unknown dynamics. The randomness of their output is a major advantage for the sim-to-real problem as stochastic simulators show better result in the policies transferability they produce. Finally, because the raw training data is stored and is compared to when using the model for prediction, the GP do not need to store the information of the data structure in its parameters like is doing a Neural Network for instance, making them extremely sample-efficient and thus perfect for real world problems.

However, there are some drawbacks when using GPs in this context. First of all, GPs are, in essence, extremely bad at extrapolating. This suppose that the input space of the GP must be widely explored during the real-world data collection, which is not always easy to do. Also, modern simulators the framework proposed here suppose that the transition function of the simulator is accessible and tunable, which is not straightforward for modern descriptive simulator such as MujoCo [56].

Acknowledgments

We would like to thank sincerely Soma Karthik for his numerous feedbacks, sharp analysis and always relevant suggestions.

CHAPTER 5 CONCLUSION

This master's degree's project was to study and propose a method for improving the Sim-to-Real transfer of RL policies on real-world robots. More precisely, it focused on Real-to-Sim, a generic idea of leveraging real-world data to improve and augment the accuracy of the simulation used to train the policies in first place. Closing the GaP: Gaussian Processes enhanced simulation for Reinforcement Learning policy transfer, proposed a method to perform such improvement.

5.1 Summary of Works

In this thesis, we investigated the use of Gaussian Processes (Gaussian Process (GP)s) to bridge the reality gap between a simulator and a real-world environment. Through two distinct experiments, one entirely in simulation as a toy case and proof of concept and the other on a robot with real-world constraints, we demonstrated that GPs can effectively estimate the discrepancy between the simulator's forward dynamics model and an unknown target environment behavior. By integrating this correction term at each step of the simulation, we showed that the generated trajectories were more accurately reflecting reality, reducing the reality gap from a behavior perspective.

Furthermore, we demonstrated that training Reinforcement Learning RL policies on this GP-enhanced simulator resulted in successful policy transfers that achieved near-optimal performance when deployed in the real world. The key to this success lies in the inherent properties of GPs: their non-parametric nature makes them highly suitable for modeling unknown dynamics as opposed to other Sim-to-Real transfer such as domain randomization, that rely heavily on the hypothesis that the simulator equations presents, even if inaccurate, a satisfying form to model the world dynamics.

Not only the use of GPs allowed the reduce of the reality gap in terms of trajectories, but we showed also that the inherent stochasticity of the GPs predictions contributes positively to Sim-to-Real transfer by generating robust policies. Because the GP output a Gaussian distribution with a mean and variance, the uncertainty of the model, optimized from training data, and propagated in the environment when sampling from those distribution to correct the simulation output, acts from a conceptual point of view, just as domain randomization, by injecting stochasticity in the environment. The policies trained in such enhanced environments are forced to be more conservative, making them more robust to the Sim-to-Real transfer. Additionally, due to their design, GPs does not need during training to understand the whole relation between input and output values as this information stays in the training set. The GP only need to grasp the impact of the closeness of two inputs on their respective outputs correlation, being then able to produce a new input on a test point by comparing it to the training inputs. Neural Networks, on the contrary, need to store all the information the data is representing in its weights, creating latent representations that it will be able to use during inference. This major difference makes the GPs extremely sample efficient which is a highly desirable property for real-world applications where data collection is expensive or constrained.

5.2 Limitations

Despite these promising results, several limitations were identified in the application of GPs for sim-to-real transfer.

A fundamental drawback of GPs is their poor extrapolation capability as the predicted mean far from training data tends to go back to prior mean, which is often taken without any knowledge to 0. This requires that the GP input space is sufficiently explored during data collection, a condition that is not always easy to fulfill in real-world scenarios and that need some task-dependent design. However, for some cases, like we showed with the CartPole, any policy can perform this data collection, which makes it easier since we do not need to already have a performing controller to sample for the GP training.

Because the output of the GP is injected in the simulator prediction, having a very high uncertainty in the prediction can lead to unstable simulator behavior making the task impossible to solve in simulation and killing the process. This makes the success of the process dependent on the quality of the hardware sensors at disposal since there can be a high level of sensor noise on the collected data, resulting in a high uncertainty of the GP prediction.

Moreover, the approach proposed in this thesis assumes that the simulator's transition function is accessible and modifiable in order to integrate the learned correction. This assumption holds for simple, analytic simulators but becomes problematic in the case of modern, descriptive simulation engines such as MuJoCo, where the underlying physics engine is not easily accessible or tunable. These factors limit the direct applicability of the method to more complex or commercially available simulators.

Because we should assume conceptually that we don't have a performing policy for the real robot before the GP training (otherwise, why are we doing this?), a concern that comes in mind is the safety when deploying the initial source policy on the robot for the first time.

Indeed, in case the system is critical, let's say an aerial vehicle, one need some guarantees of safety before being able to deploy a policy.

5.3 Future Research

To keep going with this work, the limitations that are presented in the previous sections must be met in some way. A promising direction lies in an active learning strategies to guide real-world data collection efficiently ensuring high coverage of the input space with minimal effort, compounding with the lack of extrapolation of GPs. To do so, one could rely on the uncertainty of the GPs prediction.

Concerning the dependency on sensor data, in some case the real environment is barely stochastic and all the noise comes from the sensor accuracy or the estimation that are made (take the pose estimator of a flight controller for instance), we imagined a setup in which the GP is not used for the simulation part which can create, as we said, unstable behaviors, but instead to "correct" the observation. We would then leverage the understanding of the GP of noisy data to simulate the noisy sensors in simulation, not making it more realistic in its internal state but in the perception the agent has of it.

As the safe transfer of RL policies was tackled by M. Guerrier, a co-author of the paper presented in this thesis, in [48], a natural follow-up to this work would be to merge those ideas in order to produce a complete Sim-to-Real-to-Sim loop that would allow the deployment of the presented method on every robot without the safety concern.

Another project that time prevented us from exploring deeper is the idea of using several GPs in a "chained" fashion. The idea is that where the first GP still correct the error of the simulator, we add a second GP behind to correct the error of the enhanced simulator and so on. The intuition with this is that, essentially, GP are learning the characteristic lengthscales of the dynamics relatively to the inputs. In the case of a system that shows several unmodeled dynamics that acts at different scales, this "chain" of GPs would allow the first one to account for the dynamic with the bigger scale and the others higher precision, yet with less effect, dynamics.

The process presented here close the sim-to-real-to-sim loop by enabling a first sim-to-real transfer, a real-to-sim improvement of the simulation and finally a sim-to-real transfer. Once this loop is closed, we cannot ignore the possibility to imagine an iterative process where this loop is used several times, for more complex, or more critical tasks that would need a progressive discovery of the environment for instance.

REFERENCES

[1] R. S. Sutton and A. Barto, *Reinforcement learning: an introduction*, second edition ed., ser. Adaptive computation and machine learning. Cambridge, Massachusetts London, England: The MIT Press, 2020.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1509.02971

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

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[7] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *CoRR*, vol. abs/1707.02286, 2017. [Online]. Available: http://arxiv.org/abs/1707.02286

[8] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *CoRR*, vol. abs/1806.10293, 2018. [Online]. Available: http://arxiv.org/abs/1806.10293

[9] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, “Champion-level drone racing using deep reinforcement learning,” *Nature*, vol. 620, no. 7976, pp. 982–987, Aug. 2023. [Online]. Available: https://www.nature.com/articles/s41586-023-06419-4

[10] J. Collins, D. Howard, and J. Leitner, “Quantifying the reality gap in robotic manipulation tasks,” *CoRR*, vol. abs/1811.01484, 2018. [Online]. Available: http://arxiv.org/abs/1811.01484

[11] W. Zhao, J. P. Queralta, and T. Westerlund, “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey,” in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2020, pp. 737–744, arXiv:2009.13303 [cs]. [Online]. Available: http://arxiv.org/abs/2009.13303

[12] H. Ju, R. Juan, R. Gomez, K. Nakamura, and G. Li, “Transferring policy of deep reinforcement learning from simulation to reality for robotics,” *Nature Machine Intelligence*, vol. 4, no. 12, pp. 1077–1087, Dec. 2022. [Online]. Available: https://www.nature.com/articles/s42256-022-00573-6

[13] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, “Robot Learning From Randomized Simulations: A Review,” *Frontiers in Robotics and AI*, vol. 9, p. 799893, Apr. 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2022.799893/full

[14] M. Mozian, J. Camilo Gamboa Higuera, D. Meger, and G. Dudek, “Learning Domain Randomization Distributions for Training Robust Locomotion Policies,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 6112–6117. [Online]. Available: https://ieeexplore.ieee.org/document/9341019/

[15] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[16] O. Levy and L. Wolf, “Live Repetition Counting,” in *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE, Dec. 2015, pp. 3020–3028. [Online]. Available: http://ieeexplore.ieee.org/document/7410703/

[17] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 3803–3810, arXiv:1710.06537 [cs]. [Online]. Available: http://arxiv.org/abs/1710.06537

[18] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, “Active Domain Randomization.”

[19] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, “Learning invariant feature spaces to transfer skills with reinforcement learning,” *CoRR*, vol. abs/1703.02949, 2017. [Online]. Available: http://arxiv.org/abs/1703.02949

[20] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, “Epopt: Learning robust neural network policies using model ensembles,” *CoRR*, vol. abs/1610.01283, 2016. [Online]. Available: http://arxiv.org/abs/1610.01283

[21] G. C. Goodwin and R. L. Payne, *Dynamic system identification: experiment design and data analysis*, ser. Mathematics in science and engineering. New York: Academic Press, 1977, no. v 136.

[22] P. Chang and T. Padir, “Sim2Real2Sim: Bridging the Gap Between Simulation and Real-World in Flexible Object Manipulation,” Feb. 2020, arXiv:2002.02538 [cs]. [Online]. Available: http://arxiv.org/abs/2002.02538

[23] S. Chen, S. A. Billings, and P. Grant, “Non-linear system identification using neural networks,” *International journal of control*, vol. 51, no. 6, pp. 1191–1214, 1990.

[24] K. S. Narendra and K. Parthasarathy, “Identification and control of dynamical systems using neural networks,” *IEEE Transactions on neural networks*, vol. 1, no. 1, pp. 4–27, 1990.

[25] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, “Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience,” Mar. 2019, arXiv:1810.05687 [cs]. [Online]. Available: http://arxiv.org/abs/1810.05687

[26] G. Ji, Q. Gao, Y. Xiao, and Z. Sun, “Efficient Real2Sim2Real of Continuum Robots Using Deep Reinforcement Learning With Koopman Operator,” *IEEE Transactions on Industrial Electronics*, pp. 1–11, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10875033/

[27] D. Ha and J. Schmidhuber, "World models," *CoRR*, vol. abs/1803.10122, 2018. [Online]. Available: http://arxiv.org/abs/1803.10122

[28] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots," May 2018, arXiv:1804.10332 [cs]. [Online]. Available: http://arxiv.org/abs/1804.10332

[29] N. Sontakke, H. Chae, S. Lee, T. Huang, D. Hong, and S. Hal, "Residual physics learning and system identification for sim-to-real transfer of policies on buoyancy assisted legged robots," 10 2023, pp. 392–399.

[30] A. O'Hagan, "Curve fitting and optimal design for prediction," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 40, no. 1, pp. 1–42, 1978. [Online]. Available: http://www.jstor.org/stable/2984861

[31] C. K. I. Williams and C. E. Rasmussen, "Gaussian Processes for Regression."

[32] C. E. Rasmussen, "EVALUATION OF GAUSSIAN PROCESSES AND OTHER METHODS FOR NON-LINEAR REGRESSION."

[33] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning, the MIT Press.* Massachusetts Institute of Technology, 2006. [Online]. Available: www.GaussianProcess.org/gpml

[34] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms.* USA: Cambridge University Press, 2002.

[35] ——, *Bayesian Methods for Backpropagation Networks.* New York, NY: Springer New York, 1996, pp. 211–254. [Online]. Available: https://doi.org/10.1007/978-1-4612-0723-8_6

[36] D. K. Duvenaud, "Automatic Model Construction with Gaussian Processes."

[37] M. Kuss and C. E. Rasmussen, "Gaussian Processes in Reinforcement Learning."

[38] A. Rottmann and W. Burgard, "Adaptive autonomous control using online value iteration with gaussian processes," in *2009 IEEE International Conference on Robotics and Automation.* Kobe: IEEE, May 2009, pp. 2106–2111. [Online]. Available: http://ieeexplore.ieee.org/document/5152660/

[39] A. Girard, C. E. Rasmussen, J. Q. Candela, and R. Murray-Smith, "Gaussian Process Priors with Uncertain Inputs Application to Multiple-Step Ahead Time Series Forecasting."

[40] H. S. A. Yu, D. Yao, C. Zimmer, M. Toussaint, and D. Nguyen-Tuong, "Active Learning in Gaussian Process State Space Model," Jul. 2021, arXiv:2108.00819 [cs]. [Online]. Available: http://arxiv.org/abs/2108.00819

[41] M. Buisson-Fenet, F. Solowjow, and S. Trimpe, "Actively Learning Gaussian Process Dynamics," Apr. 2020, arXiv:1911.09946 [cs]. [Online]. Available: http://arxiv.org/abs/1911.09946

[42] A. Capone, J. Umlauft, T. Beckers, A. Lederer, and S. Hirche, "Localized active learning of Gaussian process state space models," Jun. 2020, arXiv:2005.02191 [cs]. [Online]. Available: http://arxiv.org/abs/2005.02191

[43] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0005109889900022

[44] A. Girard, C. E. Rasmussen, and R. Murray-Smith, "Multiple-step ahead prediction for non linear dynamic systems – A Gaussian Process treatment with propagation of the uncertainty."

[45] J. Ko, D. J. Klein, D. Fox, and D. Haehnel, "Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. Rome, Italy: IEEE, Apr. 2007, pp. 742–747, iSSN: 1050-4729. [Online]. Available: http://ieeexplore.ieee.org/document/4209179/

[46] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[47] R. Florian, "Correct equations for the dynamics of the cart-pole system," 08 2005.

[48] M. Guerrier, S. Karthik, H. Fouad, and G. Beltrame, "Guided by guardrails: Control barrier functions as safety instructors for robotic learning," 05 2025.

[49] AgileX, "Limo," https://global.agilex.ai/products/limo-pro.

[50] J.-S. Zhao, Z.-J. Liu, and J. Dai, "Design of an ackermann type steering mechanism," *Journal of Mechanical Engineering Science*, vol. 227, 11 2013.

[51] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[52] F. Agostinelli, S. McAleer, A. Shmakov, and P. Baldi, "Solving the Rubik's cube with deep reinforcement learning and search," *Nature Machine Intelligence*, vol. 1, no. 8, pp. 356–363, Aug. 2019. [Online]. Available: https://doi.org/10.1038/s42256-019-0070-z

[53] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," Oct. 2018, arXiv:1506.02438 [cs]. [Online]. Available: http://arxiv.org/abs/1506.02438

[54] E. Heiden, D. Millard, and G. S. Sukhatme, "Real2Sim Transfer using Differentiable Physics."

[55] J. Richens, D. Abel, A. Bellot, and T. Everitt, "General agents need world models," *arXiv preprint arXiv:2506.01622*, 2025.

[56] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[57] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.

[58] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[59] J. Eschmann, D. Albani, and G. Loianno, "Learning to fly in seconds," 2024. [Online]. Available: https://arxiv.org/abs/2311.13081

[60] J. Ko, D. J. Klein, D. Fox, and D. Haehnel, "Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. Rome, Italy: IEEE, Apr. 2007, pp. 742–747, iSSN: 1050-4729. [Online]. Available: http://ieeexplore.ieee.org/document/4209179/

[61] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html