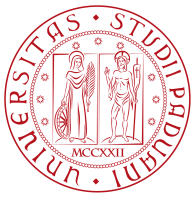

Università degli Studi di Padova

Department of Information Engineering

Master Thesis in **Automation Engineering**

System Identification meets Reinforcement Learning: probabilistic dynamics for regularization

Supervisor
Prof. Alessandro Chiuso
Università di Padova

Master Candidate
Francesco Zanini

9 September 2019
Academic Year 2018/2019

Frivolous theorem of Arithmetic:

Almost all natural numbers are very, very, very large.

Abstract

Reinforcement Learning is one of the most active research areas in the scientific community, having deep links with machine learning, optimal control and dynamic programming.
These connections emerged recently, while in the past the control community was mostly unrelated to artificial intelligence. This brought to the development of two main paradigm which both address the same problem of exploiting in the best way some knowledge on the behaviour of a system in the past to enhance future manipulation of that system: model-based and model-free perspective.
Model-free methods search directly the best behaviour over policies, acting in some way on the system and then improving their choices.
Model-based strategies tackle the same problem by seeking a good representation of the environment and then solving an optimal control problem.
In this Thesis a mixed approach is proposed, in which the interactions with the real system are carried out in both ways: a rough model is retrieved in order to play the role of a regularizer, while the punctual estimation over specific values of the policy parameter is placing reliable punctual estimates that should be fitted by the reconstructed function.

Acknowledgments

This Thesis represents the last step of my path as a student at University of Padua.
I feel lucky to have found here such a dynamic and stimulating environment, which gave me the opportunity to meet great people, whom I wanted to thank.

First of all, I would like to thank all the Professors who have contributed to my educational growth, with their courses and exams.

Above all, I must thank Prof. Alessandro Chiuso, who has gone far beyond his normal duties as supervisor, helping me through many issues and devoting much of his time to me. I really appreciated his comments, which were always appropriate and smart, and the experience he was able to bring into this work.
Luca Zancato, Ph.D. student, also deserves a mention. He supported me with his opinions and pieces of advice, being very kind and friendly from the moment I met him.

I would also like to thank my friends and colleagues and everyone who has been there for me, especially in the recent period.

I must express my profound gratitude to my parents Massimo and Luisa, who supported me financially and also allowed me to focus as much as possible on my studies, assisting me in many daily problems to save me time.

A special thank goes to my brother Giacomo: having a close person with whom you can talk daily about anything, from mathematical details to videogames, has been a fundamental resource.
I hope I can help him in the future as much as he did for me.

Nonetheless, my greatest and deepest thanks goes to the person who taught me more than anyone else, and without whom I would never have been able to reach this important milestone.

Contents

Listing of figures

Try again. Fail again. Fail better.

Samuel Beckett

1 Introduction

Reinforcement learning has gradually become one of the most active research areas in machine learning, artificial intelligence, and neural network research. The field has developed strong mathematical foundations and impressive applications. The computational study of reinforcement learning is now a large field, with hundreds of active researchers around the world in diverse disciplines such as psychology, control theory, artificial intelligence, and neuroscience. Particularly important have been the contributions establishing and developing the relationships to the theory of optimal control and dynamic programming. The overall problem of learning from interaction to achieve goals is still far from being solved, but our understanding of it has improved significantly in recent years.

The early history of reinforcement learning has two main threads, both long and rich, that were pursued independently. One thread concerns learning by trial and error that started in the psychology of animal learning. This thread runs through some of the earliest work in artificial intelligence. The other thread concerns the problem of optimal control and its solution using value functions and dynamic programming. For the most part, this thread did not involve learning.

Connections between optimal control and dynamic programming, on the one hand, and learning, on the other, were slow to be recognized. However, from

the advancements in Machine Learning the extensive use of the Markov Decision Process framework emerged some analogies that made clear a strong correspondence between the two approaches.
These two threads both came into the world of Reinforcement Learning, giving birth to two different approaches: Model-Based Reinforcement Learning and Model-Free Reinforcement Learning.
This thesis is intended to represent another step forward in the unification of the two approaches, proposing indeed a mixed strategy which exploits both perspectives.

1.0.1 Outline of the Thesis

In Chapter 2 the basic notions of Reinforcement Learning are presented, with particular attention to the Markov Decision Process framework.
In Chapter 3 two of the main strategies in System Identification are introduced, Gaussian process models and kernel-based PEM estimate, thate are necessary tools to carry out the RL procedure proposed later on.
In Chapter 4 the RL framework is converted into an optimal control problem, moreover the state of the art for both model-based and model-free approaches is presented.
Chapter 5 is the core of this work, where it is proposed the mixed approach. Initially a first theoretical attempt in deriving the regularizer is discussed, then the main algorithm adn its benefits are reviewed.
Eventually in Chapter 7 final considerations are drawn and possible future extensions of this work are described.

2
Reinforcement Learning

Reinforcement Learning is currently one of the most active research topics in the scientific field: it has recently grown very fast thanks to its interdisciplinary nature, having profound links with Artificial Intelligence, Machine Learning, Control Theory and System Identification.
These connections have stimulated the interest of the community in this discipline, which considers the whole problem of interacting with an unknown environment, benefiting from the latest improvements in aforementioned linked areas.
Nowadays Reinforcement Learning is developing both from a practical and a theoretical point of view, from implementations of algorithms working in real time, to new approaches to better understand how to refine current studies.
This chapter is focused on surveying some background notions that are necessary to comprehend the core of this work.

2.1 Introduction

Reinforcement Learning is the way in which humans, and all other animals in general, actually learn. Many of the central algorithms in this discipline

have indeed been inspired by biological learning systems.
In general, *learning* is defined as the process of acquiring new, or modifying existing, knowledge, behaviour, skills, values or preferences. Humans learn before birth and continue until death as a consequence of ongoing interactions between people and their environment: this ability has allowed a rapid evolution and the development of new solutions to problems encountered in life, in order to reach any kind of goals. Indeed the learning process is often directed at improving the way tasks are performed by the learner, from simple walking or sourcing food to driving a car or playing chess.
Different tasks yields different approaches in order to efficiently learn how to achieve a goal that has been set. Adult human beings have a clear way of conveying information, and thus they are able to help the learner by providing the required knowledge with the aim of achieving the learning goal: this is the case of a teacher-student relationship, for example. However many times in nature this ability to communicate is missing, leaving the learner alone and with no information on how to address the task it is supposed to accomplish. Indeed, for most of the terrestrial moving animals, walking and running are primary skills compared to communicating with their own peers, therefore they must face the problem of understanding how to correctly move without an explicit teacher. They might figure out something by observing how other animals behave, but the core of their modus operandi of learning is trying and trying again until they find an effective way of moving their bodies (and it is indeed a successful approach: a gazelle calf struggles to its feet minutes after being born; half an hour later it is running at 32 kilometres per hour). In general, when someone has to accomplish a specific task while operating in an environment that is at least partially unknown or uncertain, the fundamental way in which it can fulfil its assignment is by *learning from interaction*.
The uncertainty of the environment is a key feature of a Reinforcement Learning framework: if the subject operates under a fully known environment, thus having complete knowledge of all consequences of every actions relevant to the task, it can deterministically choose the best working paradigm that makes it reach its goal in the most efficient way, and no learning is involved at all.

When instead the agent is unaware of some fundamental knowledge about the scenario it is facing, it is impossible to determine even a suboptimal sequence of action that allow to solve the problem, and interaction with the environment becomes necessary.
The very first idea of learning is indeed based on trying something, observing the result, then trying something else that is supposed to work better.

2.2 Introductory Examples

The ideas described so far seem very vague and not easy to be formalized: this properly reflects the breadth of cases to which this discipline is applicable. In order to better understand the Reinforcement Learning framework and to find out the main issues encountered in this kind of problems it is necessary to take a closer look through these two simple examples.

- Consider a student in Automation Engineering who needs to write his Master Thesis using a computer which is running a text editor. He is fully aware of the mapping between buttons of the keyboard and letters appearing on the screen, thus he can carry out his task without resorting to learning (supposing he knows exactly what to write). As it has been said earlier, if everything is well known in advance, it is possible to precisely plan a sequence of actions surely leading to the completion of the objective.
  However, if the student were working on a computer with a keyboard without labels on its buttons, things would be harder. The most instinctive way of progress with the Thesis would be to press a button, see what letter appears on the screen, and to label the button accordingly.
  Once he has fully determined all labels of the keyboard he has built a reliable model of the environment, so he can go ahead carefully planning what buttons to press.
  Given the faculty that this student is enrolled in, one can expect that memory may help him retrieving the button labels of the keyboard that he needs to write down his work. This kind of prior knowledge is obviously still built relying on previous experiments on the environment, but allows to pursue a refined approach in which not all buttons

need to be pressed in order to be identified.
Moreover, some letters are used more often than others, so the student's memory will be more accurate for certain labels than others. How many and which letters does he need to test, in order to have a sufficiently precise model to complete his work, without wasting time in labelling unnecessary letters?
As a further matter, if throughout the whole work that he is writing he will never use a specific letter, it is superfluous to gain knowledge of that label: this shows again the inadequacy of the very first approach.

- A beginner archer needs to hit a target with his bow and arrows. The closer to the centre the dart sticks, the better the shooting is evaluated.
  It is not that easy for the bowman to compute exactly the strength he needs to apply to his bow in order to shoot the arrow strong enough to reach the target. As the environment is not fully known (the air resistance, the exact angle of the bow, the exact amount of strength is developed by the bow), so are the consequences of his actions, therefore there may be the need for trying to act and observe the response of the environment. If possible this can also help him build a mapping from actions to responses, relying on reactions obtained before, to have a more general (though still imprecise) idea of the environment faced.
  Hence he decides to shoot a few preliminary arrows in order to have a better idea of how to hit the centre: in the process, he is helped by the a priori knowledge of elementary physics which tells him that the tension of the bow is directly proportional to the distance that the arrow will travel.
  After some trials, he retrieved the correct way of striking arrows to maximize his score. However the following day is windy, and this affects the range of the arrows shot, thus making the retrieved model of the tension-distance function incorrect. As the force of the wind is utterly unpredictable, what is the best behaviour in striking arrows in order to hit the target even in presence of changing in direction of the wind?
  And how he can adapt its learning strategy when dealing with a target in a different position, or that is moving?

These examples show some questions that may arise in the study of a Reinforcement Learning algorithm.
Furthermore, it is clear how the interaction with the uncertain environment

plays a fundamental role in carrying out the given task. In general, having the ability to understand the consequences of actions produces a wealth of information about cause-effect relationship governing the environment, resulting in a better decision-making for fulfilling the objective.
From the perspective of Machine Learning and Control Theory, "actions" are precise inputs for machines that should accomplish a predetermined task formulated on the basis of their state. These are not fully aware of their input-output relation, but they are capable of evaluating the cost (or reward) of the results of their experience.

2.3 Key Elements of RL

Clearly the two main actors in a Reinforcement Learning paradigm are the agent and the environment, and how the former interact with the latter to gain a clearer picture of its own behaviour. However, starting to consider a more formal framework, four other distinctive elements can be identified, which play an essential role:

- The *policy* determines how the agent behaves at a given time. It is a mapping from perceived states of the environment to actions that can be taken in a specific state. The policy is the true core of a Reinforcement Learning algorithm, determining the agent's behaviour and thus its performance over the assignment. It can be both deterministic or stochastic. From a Control Theory point of view the policy is often regarded as a function depending both on the state and on the control parameters, and providing the input control.

- The *reward signal* (or *cost*) defines the goal of a Reinforcement Learning problem. This signal is a scalar measure of the goodness of the state in which the agent is located. Maximizing the reward is the only objective for the agent, and it is the primary basis for altering the policy: if the action selected is followed by a low reward, the policy may be changed in order to choose a different action with a higher reward. The aim of the agent can be defined by either setting a reward function or a cost function. Clearly in the latter case the agent will try to minimize the signal from the environment. A classical example for a cost

function could be the distance of a moving robot from a predetermined desired locus.

- The *value function* helps the agent to have a long term view of the cost it will face following a certain policy. The *value* of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas the rewards determine the immediate desirability of environmental states, values indicate the long-term appealing of the state on the basis of the states that are likely to follow and their rewards. Recalling the previous example, if the moving robot has adopted a certain policy that brings it in an advantageous state for the next time step but leads it in subsequent states with law rewards, the reward at the current time will be high but the value function will be law.

- The *model of the environment* allows the agent to make inferences about how the system will behave in response of a certain action, by mimicking the behaviour of the environment. Given a state and an action, the agent can predict the next state and reward by relying on the knowledge of the model, and therefore decide on the course of action without experiencing it. This is a key element in model-based Reinforcement Learning, in which indeed the best policy is chosen through *planning*, as will be further explained in Chapter 4, Section 4.3.

Rewards are in a sense primary, whereas values, as prediction of rewards, are secondary. Indeed without rewards there could be no values, and the only purpose of estimating values is achieving more reward. Nonetheless, the choice of the policy is made according to values rather than rewards. By seeking actions that yield high values, it is more likely to follow policies which guarantee high reward over the whole run, as it is well illustrated by the maze example in Section 2.5.
However, while rewards are directly provided by the environment, values need to be estimated from sequence of observations of the agents. This is one of the key feature of the Reinforcement Learning paradigm.

2.4 Reinforcement Learning

Reinforcement Learning is a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives while interacting with a complex, uncertain environment.
The learner is not told which actions to take, but instead must discover which actions yield the highest reward by trying them. Indeed in most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation, affecting all future costs as well.
Reinforcement Learning explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment, in all its aspects. This one of the main difference with *Machine Learning*, that instead addresses the more isolated sub-problem of generalizing information coming from a training set, or finding hidden structures in a given dataset.
In *Supervised learning* the learner is fed with labelled examples from a knowledgable external supervisor: each samples could be a description of a situation together with a specification of the correct action to take, the label. The goal of the learner is to extrapolate the provided information in order to choose the right action even in situations not present in the training set. This kind of learning is not suitable for learning from interaction, since in the prescribed framework there is no helpful teacher and the learner should learn from its on experience.
Unsupervised learning is about finding structures hidden in collection of unlabelled data. This kind of approach can be indeed helpful in the Reinforcement Learning paradigm: uncovering structures in an agent's experience may lead to build a good model of the environment, but this does not solve the problem of maximizing the reward signal, or minimizing the cost.
Reinforcement Learning is instead concerned with an agent that has an explicit goal, can sense aspects of its environment, and can choose actions to affect it.
As it came out of the introductory examples, a key feature of this framework is the trade-off between *exploration* and *exploitation*. By exploring different actions on the environment, it is possible to build a more complete and accurate model of its behaviour, although it is a costly and time-consuming

operation. Moreover the process might lead to perform actions that have a low reward. To obtain a lot of reward, a Reinforcement Learning agent should prefer actions that it has tried in the past and found to be effective in producing reward. But in order to discover those actions, it must resort to exploration, that is trying action not selected before.
The agent has to exploit what it has already experienced to have a safe and high reward, but it has also the need to explore different policies in order to make better action selection in the future. Neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favour those that appears to be best.
Things gets more involved if the output of the environment is not deterministic, so the reward is a random variable as well. This allows the same action to perform both good or bad in different time instants: a single action must be then tested several times in order to get a reliable estimate of its expected reward.
Being such an interdisciplinary subject, the concept of *state* in Reinforcement learning is not bound to represent the physical position of a moving robot, but it can be seen as whatever information is available to the agent about the environment it is facing. It is assumed that the state signal is produced by some preprocessing system that is nominally part of the agent's environment. The state plays a fundamental role as it represent both the input and the output of the model, and also a variable on which both cost and value function depend.

2.5 A First Formalization

The problem of Reinforcement Learning is here formalized using ideas from Dynamical System Theory and Markov Decision Processes.
A learning agent must be able to sense the state in which it is located an must be able to take actions which affect that state, having a well defined goal expressed by the reward. The reward itself is a function of the state, therefore different actions lead to different rewards.

The agent and the environment interacts continually, the former selecting actions and the latter responding to them giving rise to a reward, and presenting new situations to the agent.

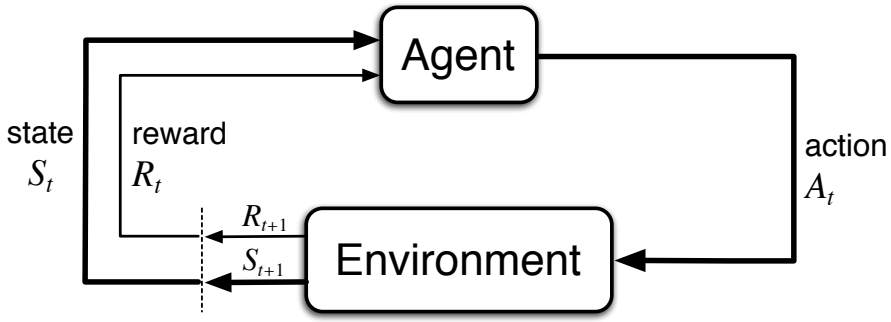

Figure 2.1: The agent-environment interaction scheme represented through a Markov Decision Process: the agent act on the environment that provides both a reward signal and a new state for the agent.

The agent and the environment interact at each of a sequence of discrete time steps, $t, t+1, \ldots$, in which the agent receives some representation of the environment's state $S_t \in \mathcal{S}$ and based on that, it select an action $A_t \in \mathcal{A}(S_t)$. As a consequence, one time step later the agent receives the reward signal $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and find itself in a new state S_{t+1}.
Being the environment uncertain, the results of the actions R_t and S_t are random variables and have well defined probability distribution given the knowledge of the preceding state and action.
In a finite Markov Decision Process, sets of states, actions and rewards (respectively $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$) all have a finite number of elements. Hence for a particular values of the random variables $s' \in \mathcal{S}$ and $r \in \mathcal{R}$ there is a probability of those values occurring at time t, given specific values of preceding states and actions:

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{2.1}$$

for all $s, s' \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in \mathcal{A}$.
In a Markov Decision Process the probabilities given by $p(\dot{} \mid \dot{})$ completely characterize the environment dynamics, that indeed depends only on the preceding action and state. The latter condensates all the information on

previous interactions between agent and environment that affect the future: this feature of the state is indeed called *Markov property*.
From the four-arguments dynamics function p, it is possible to compute anything else one might want to know about the environment, such as state-transition probabilities $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$:

$$p(s' \mid s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a) \tag{2.2}$$

The expected rewards for state-action pairs $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{2.3}$$

The expected reward for state-action-next-state triples $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)} \tag{2.4}$$

The MPD framework is abstract and flexible and can be applied to many different problems in many different ways. Recall the introductory examples: actions cab be both low-level controls or high-level decisions. Similarly, the states can take a wide variety of forms.
One interesting feature of the Reinforcement Learning paradigm is that the boundary between agent and environment is typically different from the physical boundary of a robot's or animal's body. This of course depends on the application for which the algorithm is designed, but often this partition is set closer to the agent, leaving to the environment also elements that are commonly thought as part of the agent.
Consider for example a robotic arm with n joints: the motors, the mechanical linkages, and its sensing hardware should usually be considered part of the environment rather than the agent. The actions of the agent might be a n-tuple indicating the position of each joint and the task to be accomplished may be to bring the end-effector in a specific position. Therefore the agent should learn what is the best set of values by trying different tuples and observing the position of the end effector, or even solely its distance from

the goal. By defining the cost as the distance of the end effector from the desired position, it is possible to see that the sensor giving this information must be part of the environment and not part of the agent, since the cost should come from the former.
The general rule is that anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of the environment. Surely the reward computation should be something on which the agent has no full control, hence it must be external to the agent, since it defines the task that has to be accomplished and it would be meaningless if the agent could change it arbitrarily. Nonetheless, the agent may have knowledge about the environment and how its reward are computed as a function of the states and actions taken, but still facing a challenging Reinforcement Learning problem: a classical example is given by the maze. If the person trapped in the labyrinth periodically hears a bell ringing exactly on the exit, he may have an idea of how far or near the target is. It knows everything about the environment: how to move from one point to another is straightforward and the distance from the objective is given by the ringing bell, thus it should be easy to get out. However the path leading to the exit is still unknown, and it might be more than likely that the route that seems to point exactly towards the goal turns out to be a dead-end road.
The expediency of shrinking the concept of agent as much as possible is useful in order to restrict the framework to nothing more than what is essential, an nothing less than everything useful.
A crucial aspect of a Reinforcement Learning algorithm is how to set the reward (or the cost). The reward should best represent what the agent is supposed to achieve as final goal, avoiding giving awards to minor success that could deeply affect the way in which the agent is learning, leading to bad performances. This is exactly what happens in the labyrinth example. In fact if the agent is supposed to learn how to exit from the maze, defining the cost as the distance between the agent and the exit is not a good idea: clearly the agent will take the path pointing towards the goal, but when reaching a dead-end road it could remain stuck. Indeed taking the same way back would result in an increase of the cost, if it does not take into account a sufficiently long time horizon, and therefore the only way

for the agent to minimize the cost would be to stand still. A better reward signal would be to assign a constant negative value for every time instant the agent is still within the maze: this express in a more accurate way the concept of wanting it outside the labyrinth.

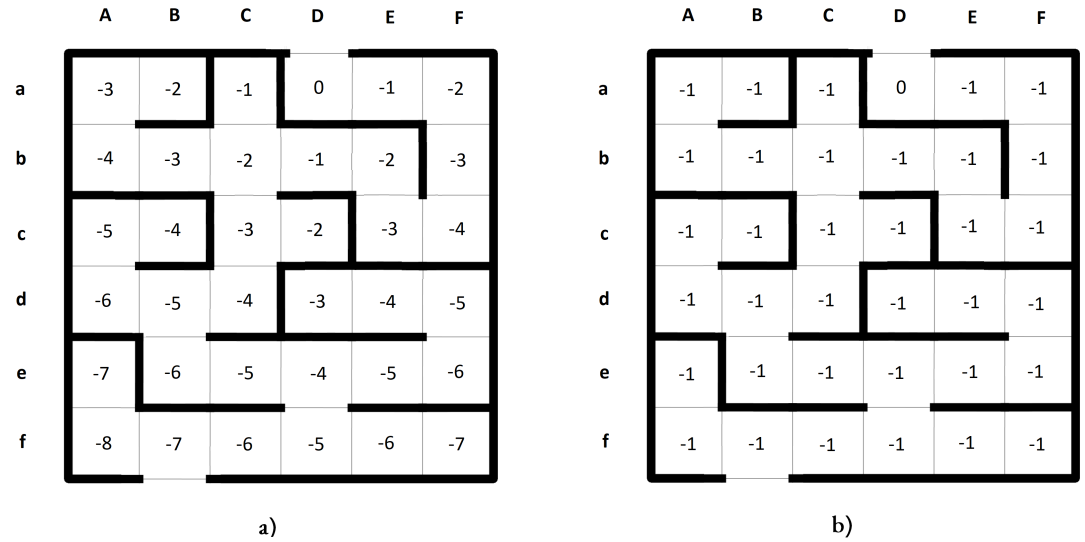

Figure 2.2: The maze example: the agent can move within the labyrinth starting from box *fB* and it needs to exit the puzzle by reaching box *aD*. At every time step thee agent can choose to move in any direction that is not obstructed by a wall. Every box is thought as a different state, in which the resultant cost is written. The figures refer to two different cost signals.
In scheme **a)** it is presented the case of the cost function defined as the distance (in fictional steps) between the position of the agent and the final goal. Even in this simple example, this choice causes some problems. Indeed the agent is likely to move quickly towards box *aC*, where it reaches a sort of local minimum for the cost function, and therefore it has no intention of leaving that box, remaining stuck. The path that would lead it to the exit goes through some boxes with high cost.
In case **b)** the only desire of exiting the maze is better expressed, so the agent is more likely to fulfil its objective by reaching the final goal. In fact here it is penalized how much "time" it spends in searching the exit, forcing it to find its way out.

Having a long-term view of the costs it will face can lead to a much better decision-making for the agent. As it has been said above, the true value that should be taken into account in choosing a policy is the value function. More formally, the agent will not try to minimize just the next reward R_{t+1}, but instead the expected return, that is a function of the reward sequence.

In the most simple case it is given by:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_{t+T} \tag{2.5}$$

for a certain time horizon $T \in \mathbb{N}$.
This approach is useful when the application has a natural definition for a final step T, or when the agent cares about only a specific horizon. When the agent-environment interaction breaks naturally into subsequence, they are called *episodes*. Episodes can all be considered to end in the same terminal state, with different rewards for different outcomes. Tasks based on episodes are called *episodic task*.
In order to extend the set in which the value of T can be chosen to $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$, it is necessary to consider also a discount factor $\gamma \in [0,\, 1]$, otherwise the return could easily be infinite as well. By taking:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_{t+T} = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{2.6}$$

it is possible to weight how much the agent cares about the future by tuning the value of γ, which is called *discount rate*. If γ is equal to 0, the agent is not concerned with planning his behaviour, but only cares about maximizing the payoff in the next step; if γ is equal yo 1, the reward at every time step counts the same.
By taking $\gamma < 1$, even if the sum in (2.6) the time horizon is infinite, i.e. $T = \infty$, the return will have a finite value as long as the reward sequence $\{R_t\}$ is bounded.
Almost all Reinforcement Learning algorithms involve the estimation of value functions, that can be seen as a measure of the goodness of a given state. A state is better than another if it yields a higher return for the considered time horizon. Clearly the subsequent rewards will depend on the actions taken by the agent, therefore value functions are defined with respect to particular ways of acting, called policies.
Formally, a policy is a mapping from states to probability distributions over the set of actions $\mathcal{A}$. If the agent is following policy π at time t, then $\pi(a \mid s)$ is the probability that $A_t = a$ if $S_t = s$.

The value function of a state s under a policy π , denoted as $v_\pi(s)$, is the expected return when starting in s and following π thereafter. For MDPs, it is possible to define $v_\pi(s)$ as:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \,|\, S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s \right] \tag{2.7}$$

for all $s \in \mathcal{S}$.
If an agent follows policy π and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value $v_\pi(s)$ as the number of times that state is encountered approaches infinity. This way of estimating value functions follows the principle of *Monte Carlo methods*: they indeed involve averaging over many random samples of the actual returns. The Monte Carlo approach will be discuss in further detail when the main contribution of this work will be presented (Chapter 5).
A fundamental property of value functions is that they satisfy recursive relationships. For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi \left[G_t \,|\, S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s \right] \\ &= \sum_a \pi(a \,|\, s) \sum_{s'} \sum_r p\left(s', r \,|\, s, a\right) \left[r + \gamma \mathbb{E} \left[G_{t+1} \,|\, S_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a \,|\, s) \sum_{s', r} p\left(s', r \,|\, s, a\right) \left[r + \gamma v_\pi\left(s'\right) \right] \end{aligned} \tag{2.8}$$

for all $s \in \mathcal{S}$.
This last equation is called *Bellman equation* for $v_\pi(s)$. It express a relationship between the value of a state and the values of its successor states, and it is a key ingredient of many ways to compute, approximate and learn v_π.
The final goal of a Reinforcement Learning algorithm is to find a policy that achieves a lot of reward over the long run. Since value function establish a partial ordering over policies, for finite MDPs it is possible to define an

optimal policy in the following way. A policy π is said to be better than or equal to a policy π' if the expected return is greater than or equal to that of π' for all states. Namely, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. There will be always at least one policy that is better than or equal to all other policies: this is an *optimal policy*, denoted as π^*.
There be more than one optimal policy, however they share the same state-value function, defined as:

$$v^*(s) \doteq \max_\pi v_\pi(s) \tag{2.9}$$

for all $s \in \mathcal{S}$.
Clearly the Reinforcement Learning agent will learn the optimal policy only rarely and in simple cases: for most of the task that are often considered, optimal policies can be generated only with extreme computational cost.

2.6 Applications

Reinforcement Learning has countless fields of applications. This can be easily understood by thinking about how many objects in every day life are used in a repetitive way, with small variations in operative behaviour over time, that could be learned automatically.
Moreover, there is a much larger set of things that could benefit from learning from interaction, although not having necessarily a repetitive task. One fascinating example is given by dynamic random access memories (DRAM). The job of a DRAM memory controller is to efficiently use the interface between the processor chip and an off-chip DRAM system to provide the high-bandwidth and low-latency data transfer necessary for high-speed program execution. A memory controller needs to deal with dynamically changing patterns of read/write requests while adhering to a large number of timing and resource constraints required by the hardware. This is a formidable scheduling problem, especially with modern processors with multiple cores sharing the same DRAM.
Considering policies that take advantage of past scheduling experience and

account for long-term consequences of scheduling decisions can improve the way the DRAM works over time for refreshing operations.
In 2008 researchers in [3] indeed designed a reinforcement learning memory controller and demonstrated that it can significantly improve the speed of program execution over what was possible with conventional controllers at the time.
One of the most suitable environment for the application of Reinforcement Learning algorithms is the game framework. Indeed game such as Checkers are characterized by a simple environment in which most of the time the set of actions for a player is quite limited and setting a reasonable reward function is often straightforward. If this is the case, the procedure of learning is computationally feasible even exploring all set of actions and considering a long time horizon in reward evaluation.
However considering just little more challenging cases such as Chess or Go, an exhaustive search becomes unfeasible because the search space grows very large. Nonetheless for Go the main difficulty is the definition of an adequate position evaluation function. A good evaluation function allows search to be truncated at a feasible depth by providing relatively easy-to-compute predictions of what deeper search would likely yield: there is still no such function for Go.
One of the greatest challenges in applying reinforcement learning to real-world problems is deciding how to represent and store value functions and/or policies. Unless the state set is finite and small enough to allow exhaustive representation by a lookup table—as in many of our illustrative examples—one must use a parametrized function approximation scheme. Whether linear or non-linear, function approximation relies on features that have to be readily accessible to the learning system and able to convey the information necessary for skilled performance. Most successful applications of reinforcement learning owe much to sets of features carefully handcrafted based on human knowledge and intuition about the specific problem to be tackled.
Function approximation becomes necessary when dealing with more complex frameworks such as Go itself or video games, which represent an interesting middle ground between a too limited framework and a real case scenario.

Another interesting feature of Reinforcement Learning is its connection with Neuroscience. Indeed many core algorithms developed for Reinforcement Learning have been discovered to actually operate inside the brain. This point of contact involves *dopamine*, a chemical deeply involved in reward processing in the brains of mammals. Dopamine appears to convey temporal-difference errors to brain structures where learning and decision making take place. From the convergence of computational reinforcement learning and results of neuroscience experiments the *reward prediction error hypothesis of dopamine neuron activity* emerged as a successful tool for thinking about reward-based learning in animals.

3

System Identification Synopsis

The generality of the concept of *system* allows the study of models associated with it to play a leading role in almost all modern scientific disciplines.
A system is defined as an object in which variables of different kind interact and produce observable signals and its model is a set of equations which describes it as sharply as possible, establishing relations among certain observed variables of the system.
In this chapter the basics of System Identification procedure are briefly surveyed, with particular attention to *Gaussian process models* and *kernel-based PEM estimate*, which are two of the most used and widespread approaches in learning dynamical systems and are part of the necessary background to follow the main part of this Master Thesis.

3.1 Basics

The system identification procedure is based on three fundamental elements:

- The data
- The model class containing the candidate model

- A choice criterium by which candidate models can be "ranked" using the data

All models assume a directional dependency between an input or covariate u and the corresponding observable output or response y. Based on empirical observation the model attempts to describe the conditional distribution $p(y\,|u)$.
Moreover, the class of models described in this chapter assumes that this relation can be decomposed into a systematic and a random component: the systematic dependency is given by a function $g : \mathbb{R}^n \to \mathbb{R}^m$ such that the sampling distribution, i.e. the likelihood, is of the form

$$p\left(y \mid g\left(u, \phi\right), \varphi\right) \tag{3.1}$$

which describes the random aspects of the data-generating process.
In models of the form (3.1) the parameter ϕ influences the core function, while φ takes care of additional variables which define the random component.
Assume to observe N times the output $\{y(1), \ldots, y(N)\}$ and the input $\{u(1), \ldots, u(N)\}$ of the considered system. Beside, $y(k) \in \mathbb{R}^m$ and $u(k) \in \mathbb{R}^n$ are taken simultaneously. This experiment can be described by the regression model:

$$y\left(k\right) = g_k\left(u\left(k\right), \phi\right) + e\left(k\right), \quad k = 1, \ldots, N \tag{3.2}$$

where $g_k : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ is known, while parameter ϕ is chosen in such a way to describe the relation between $y(k)$ and $u(k)$ properly through $g_k(\cdot, \phi)$; $e(k)$ is a random vector specified by φ and models both errors and the inadequacy of $g_k(\cdot, \phi)$ to describe the input-output relation at the k-th measurement.
Clearly for different values of the parameter ϕ, different values for the variance of the noise should be considered. Suppose for example that data were generated from a real model which is actually contained in the model class considered in (3.2), with low realizations of the noise. If the identification procedure were good enough to retrieve a suitable parameter $\bar{\phi}$, there would be no need to consider a large value for the noise variance, since data are

well explained. On the other hand a poor model needs to resort to large disturbances to account for the lack of precision in fitting the data. Therefore parameters ϕ and φ are strictly connected: this issue will be addressed later on.
Different frameworks rely on different definitions of the parameter ϕ. Indeed by taking it as an unknown but fixed vector, the estimation problem will then fall under the Fisherian paradigm, in which it is postulated that there exist an actual ϕ_0 which describes exactly the considered system. This hypothesis however is unrealistic since mathematical models are just approximations of a real system: taking instead ϕ as a random vector, it is followed the Bayesian approach.
As will be explained better in the next section, Bayesian inference is used to describe how observed data change the beliefs and uncertainties about the values of parameters in a given model, and subsequently how to address the uncertainty about the model itself and how to choose the best model to explain data.
Moreover, it is possible to further distinguish the estimation problem according to the space to which the parameter belongs, i.e. $\Phi \in \mathbb{R}^p$. When p is finite, in (3.2) it is considered a parametric family of probability densities $\mathcal{F} = \{f_\phi, \phi \in \Phi\}$ for $y(k)$ thus it is a parametric estimation; if it is considered $p = \infty$, i.e. an infinite dimensional parameter, it becomes a non-parametric estimation problem.

3.2 Gaussian Process Models

The aim of inference is to identify the systematic component $g(\cdot)$ from empirical observations and prior beliefs: data comes in the form of an input-output pairwise observations, while prior belief needs to be formalised.
The parametric approach is to assume a structure for $g(u, \phi)$ with finitely many parameters ϕ (as it happens with Neural Networks): in this case the prior uncertainty about g is usually expressed in terms of a prior distribution on ϕ. However, in situations where the form of g is not known, assuming a particular parametric form might be too restrictive: the drawback of para-

metric models is that the accuracy by which g can be identified is bounded by the best function having the assumed structure. Therefore it is an intuitively appealing approach to make inference about g directly, i.e. in a non-parametric way.

The name *Gaussian process model* refers to using a Gaussian process (GP) as a prior on g. The Gaussian process prior is non-parametric in the sense that instead of assuming a particular parametric form of $g(u, \phi)$ and making inference about ϕ, the approach is to put a prior on function values directly. Hence, the model in (3.1) now rely on a latent function g, giving:

$$p\left(y \,|\, g, \varphi\right) \tag{3.3}$$

which would be equal to $\mathcal{N}(y \,|\, g(u), \sigma^2)$ in the case of a normal noise, with the variance controlled by the additional parameter φ, i.e., $\varphi = \sigma^2$. A Gaussian process prior on g technically means that a priori the joint distribution of a collection of function values $g_{[1,N]} = [g(u_1), \ldots, g(u_N)]^T$ associated with any collection of N inputs $u_{[1,N]} = [u_1, \ldots, u_N]^T$ is multivariate normal:

$$p\left(g_{[1,N]} \,|\, u_{[1,N]}, \psi\right) = \mathcal{N}\left(g_{[1,N]} \,|\, \mu, K\right) \tag{3.4}$$

with mean μ and covariance matrix K.

A Gaussian process is then specified by a mean function $\mu(u)$ and a covariance function $k(u, u', \psi)$ such that $K_{ij} = k(u_i, u_j, \psi)$ and $\mu = [\mu(u_1), \ldots, \mu(u_N)]^T$. By choosing a particular form of covariance function we may introduce hyper-parameters ψ to the Gaussian process prior. Depending on the actual form of the covariance function $k(u, u', \psi)$ the hyper-parameters ψ can control various aspects of the Gaussian process.

Note that the sampling distribution of y in (3.3) depends on g only through $g(u)$. As an effect the likelihood of y, given the data $\mathcal{D} \doteq [u_{[1,N]}, y_{[1,N]}]$, factorises

$$p\left(y \,|\, g, \varphi\right) = \prod_{i=1}^{N} \left(y_i \,|\, g\left(u_i\right), \varphi\right) = p\left(y \,|\, g_{[1,N]}, \varphi\right) \tag{3.5}$$

and depends on g only through its value at the observed inputs $g_{[1,N]}$. According to the model, conditioning the likelihood on $g_{[1,N]}$ is equivalent to

conditioning on the full function g. This is of central importance since it allows us to make inference over finite dimensional quantities instead of handling the whole function g.
The posterior distribution of the function values, $p(g_{[1,N]} \,|\, \mathcal{D}, \varphi, \psi)$, can be computed according to the Bayes rule, so that it is possible to retrieve the posterior predictive distribution of $g(\tilde{u})$ for any input $\tilde{u}$.
Furthermore, the joint prior distribution of $g_{[1,N]}$ and $g(\tilde{u})$ due to the GP prior is multivariate normal

$$p\left(g_{[1,N]}, g\left(\tilde{u}\right) \,|\, u_{[1,N]}, \tilde{u}, \psi\right) = \mathcal{N}\left(\begin{bmatrix} g_{[1,N]} \\ g\left(\tilde{u}\right) \end{bmatrix} \,|\, \begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix}\right) \tag{3.6}$$

where the covariance matrix is partitioned such that K_{**} is the prior covariance matrix of the $g(\tilde{u})$ and K_* contains the covariances between $g_{[1,N]}$ and $g(\tilde{u})$.
The conditional distribution of $g(\tilde{u}) \,|\, g_{[1,N]}$ is then given by

$$\begin{gathered} p\left(g\left(\tilde{u}\right) \,|\, g_{[1,N]}, u_{[1,N]}, \tilde{u}, \psi\right) = \\ \mathcal{N}\left(g\left(\tilde{u}\right) \,|\, \mu_* + K_*^T K^{-1}\left(g_{[1,N]} - \mu\right), K_{**} - K_*^T K^{-1} K_*\right) \end{gathered} \tag{3.7}$$

which is again multivariate normal.

3.2.1 Regression with Normal Noise

Bayesian inference about the latent function g in Gaussian process regression models is analytically tractable if the observational noise is assumed to be normally distributed.
The model assumes $y = g(u) + \epsilon$ where the additive observational error follows a normal distribution $\mathcal{N}(\epsilon \,|\, 0, \sigma^2)$ such that $p(y \,|\, g(u), \varphi) = \mathcal{N}(y \,|\, g(u), \sigma^2)$. Again the noise variance σ^2 is an additional parameter of the likelihood.
Given the observed input locations $u_{[1,N]}$ the likelihood of g becomes

$$p\left(y \,|\, g, \varphi\right) = \mathcal{N}\left(y \,|\, g_{[1,N]}, \sigma^2 I_N\right) \tag{3.8}$$

By detrending the data, it is possible to reasonably set the prior mean function to zero, $\mu(u) = 0$. Therefore the prior on the latent function values at the observed inputs is

$$p\left(g_{[1,N]} \,|\, u_{[1,N]}, \psi\right) = \mathcal{N}\left(g_{[1,N]} \,|\, 0, K\right) \tag{3.9}$$

where the elements of the covariance matrix K are computed element-wise using a covariance function $k(u, u', \psi)$. Since likelihood and prior are both multivariate normal, the posterior on g can be calculated analytically.
Then the posterior predictive distribution of the latent function values $g(\tilde{u}_{[1,\tilde{N}]})$ for an arbitrary set of test locations $\tilde{u}_{[1,\tilde{N}]}$ takes the form

$$\begin{gathered} p\left(g(\tilde{u}_{[1,\tilde{N}]}) \,|\, \mathcal{D}, \tilde{u}_{[1,\tilde{N}]}, \varphi, \psi\right) = \\ \mathcal{N}\left(g\left(\tilde{u}\right) \,|\, K_*^T \left(K + \sigma^2 I_N\right)^{-1} y, K_{**} - K_*^T \left(K + \sigma^2 I_N\right)^{-1} K_*\right) \end{gathered} \tag{3.10}$$

The above argumentation generalises to an arbitrary set of input locations, meaning that the posterior process $g \,|\, \mathcal{D}$ is again a Gaussian process with posterior mean and covariance function

$$\begin{aligned} \mu_*\left(u\right) &= k\left(u\right)^T \left(K + \sigma^2 I_N\right)^{-1} y \\ k_*\left(u, u'\right) &= k\left(u, u'\right) - k\left(u\right)^T \left(K + \sigma^2 I_N\right)^{-1} k\left(u'\right) \end{aligned} \tag{3.11}$$

where $k(u) = [k(u_1, u), \ldots, k(u_N, u)]^T$ is a vector of prior covariances between u and the training inputs $u_{[1,N]}$. Thereby for any given set of input locations $\tilde{u}_{[1,\tilde{N}]}$ it is possible to compute the posterior predictive distribution of the corresponding function values $g(\tilde{u}_{[1,\tilde{N}]})$ which is a multivariate normal distribution.
So far inference over the latent function g has been described for a given noise variance $\varphi = \sigma^2$ and hyper-parameters ψ of the Gaussian process prior. Typically, values of these parameters are not known a priori. In a full Bayesian setting one should also perform inference over these parameters.
Assigning prior distributions $p(\varphi)$ and $p(\psi)$ it is possible to resort to maximum likelihood approaches in order to find out the best performing values

for the above variables; alternatively validations schemes are available for an efficient way of estimating those quantities.

3.3 Dynamic Models

Clearly the regression model in (3.2) is a static model because g_k describes the relation only between $y(k)$ and $u(k)$. Choosing g_k which describes the relation between $y(k)$ and $u(k), \ldots, u(k-h)$, with $h \in \mathbb{N}$, it is possible to obtain a dynamic regression model having the structure:

$$y = g(u, \phi) + e \tag{3.12}$$

where the variance of e again depends on the additional parameter φ.
However considering a more control-oriented paradigm, it arises the need to deal with a more structured model, including both the input-output relation and the effects on the input of the feedback controller.
Defining $\mathcal{F}(z)$ as the model of the plant, $\mathcal{G}(z)$ as a forming filter that shapes the noise, $-\mathcal{H}(z)$ as the controller, and $w(t)$ as the reference signal, a general linear model with feedback describing the dynamical relation between u and y is:

$$\begin{cases} y\left(t\right) = \mathcal{F}\left(z\right) u\left(t\right) + \mathcal{G}\left(z\right) e\left(t\right) \\ u\left(t\right) = \mathcal{H}\left(z\right) y\left(t\right) + r\left(t\right) \end{cases} \tag{3.13}$$

where

- $\mathcal{F}(z)$, $\mathcal{G}(z)$, $\mathcal{H}(z)$ are analytic and causal;
- $\mathcal{G}(z)$ and $\mathcal{G}(z)^{-1}$ are analytic, causal, BIBO stable and $\mathcal{G}(\infty) = I$;
- $\mathcal{F}(\infty) = 0$ (strictly causal);
- e is white noise;
- w is a *wide sense stationary* deterministic process
- $\mathcal{F}$, $\mathcal{G}$, $\mathcal{H}$ are such that the transfer matrices governing the relations $w \to y$, $e \to y$, $y \to u$ and $e \to u$ are BIBO stable.

A block diagram scheme representing the model with feedback in (3.13) is depicted in Fig. 3.1.

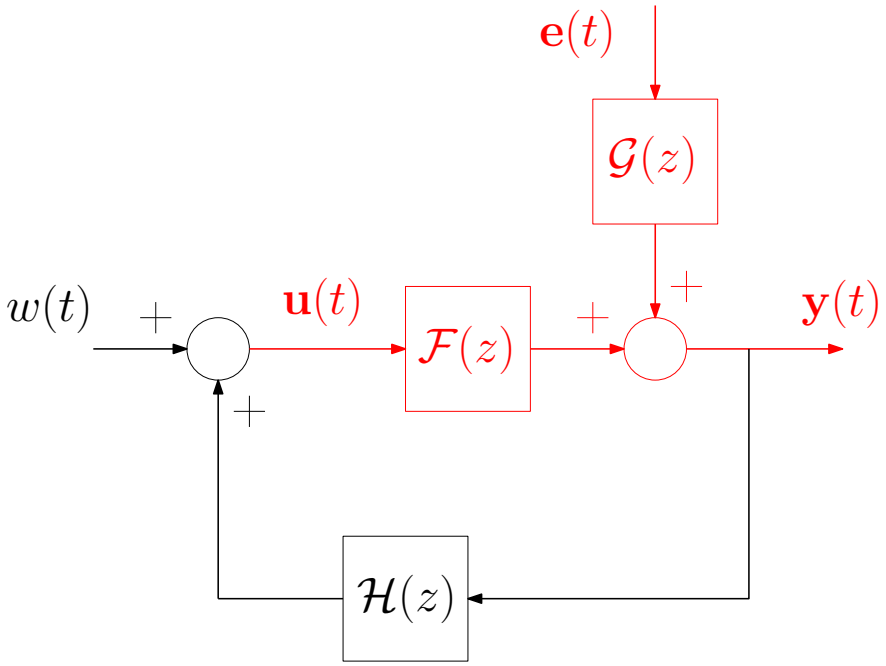

Figure 3.1: The model with feedback represented through a block diagram. The aim of System Identification is to identify the red part, i.e. the transfer function of the plant and the "shape" of the noise that models both disturbances and inadequacies of the model.

Therefore, with the aim of finding the relation between y and u through model

$$y(t) = \mathcal{F}(z)\, u(t) + \mathcal{G}(z)\, e(t) \tag{3.14}$$

it would be useful to define a parameter ϕ characterizing model (3.14).

3.3.1 Box-Jenkins Models

The Box-Jenkins model is given by:

$$y(t) = \frac{B(z)}{F(z)} u(t - n_k) + \frac{C(z)}{D(z)} e(t) \tag{3.15}$$

where

$$\begin{aligned} B(z) &= \sum_{k=0}^{n_B-1} b_k z^{-k} \rightarrow \deg(B(z)) = n_B - 1 \\ F(z) &= 1 + \sum_{k=1}^{n_F} f_k z^{-k} \rightarrow \deg(F(z)) = n_F \\ C(z) &= 1 + \sum_{k=1}^{n_C} c_k z^{-k} \rightarrow \deg(C(z)) = n_C \\ D(z) &= 1 + \sum_{k=1}^{n_D} d_k z^{-k} \rightarrow \deg(D(z)) = n_D \end{aligned} \tag{3.16}$$

and $n_k \in \mathbb{N}$ is the input delay, e is white noise with variance σ^2.
The parametrization yields:

$$\phi = [b_0, \ldots, b_{n_B-1}, f_1, \ldots, f_{n_F}, c_1, \ldots, c_{n_C}, d_1, \ldots, d_{n_D}]^T \tag{3.17}$$

The number of parameters p in ϕ is:

$$p = n_B + n_F + n_C + n_D \tag{3.18}$$

3.3.2 Output-Error Models

The Output-Error model is a particular Box-Jenkins model with $n_C = 0$ and $n_D = 0$, that is $C(z) = 1$ and $D(z) = 1$.

3.3.3 ARMAX Models

The Box-Jenkins model is given by:

$$A(z)\, y(t) = B(z)\, u(t - n_k) + C(z)\, e(t) \tag{3.19}$$

where

$$\begin{aligned} A(z) &= 1 + \sum_{k=1}^{n_A} a_k z^{-k} \rightarrow \deg(A(z)) = n_A \\ B(z) &= \sum_{k=0}^{n_B-1} b_k z^{-k} \rightarrow \deg(B(z)) = n_B - 1 \\ C(z) &= 1 + \sum_{k=1}^{n_C} c_k z^{-k} \rightarrow \deg(C(z)) = n_C \end{aligned} \tag{3.20}$$

and $n_k \in \mathbb{N}$ is the input delay, e is white noise with variance σ^2.
The parametrization yields:

$$\phi = [a_1, \ldots, a_{n_A}, b_0, \ldots, b_{n_B-1}, c_1, \ldots, c_{n_C}]^T \tag{3.21}$$

The number of parameters p in ϕ is:

$$p = n_A + n_B + n_C \tag{3.22}$$

3.3.4 ARX models

The ARX model is a particular ARMAX model with $n_C = 0$, that is $C(z) = 1$.

For every model structure considered above, Φ contains only vectors for which the conditions on the model with feedback in (3.13) are respected.

3.4 **PEM method**

The PEM method is a choice criterium for the best parameter ϕ which is the analogue of the Least-Squares principle in the static case.
Consider the model structure $\mathcal{M}$:

$$\mathcal{M}(\phi) \; : \quad y(t) = \mathcal{F}_\phi(z)\, u(t) + \mathcal{G}_\phi(z)\, e(t) \tag{3.23}$$

that can be implemented through BJ, OE, ARMAX or ARX models.
This approach rely on predictions of the output variable, therefore in the following is computed a suitable one-step-ahead predictor of y given its past and the past of u.
By multiplying on the left equation (3.14) by $\mathcal{G}(z)^{-1}$ (causal and BIBO stable by assumption), it can be written:

$$\mathcal{G}(z)^{-1} y(t) = \mathcal{G}(z)^{-1} \mathcal{F}(z) u(t) + e(t) \tag{3.24}$$

Since $\mathcal{G}(\infty) = I$, by defining

$$\mathcal{G}_1(z) \doteq \mathcal{G}(z) - I \tag{3.25}$$

it will be $\mathcal{G}_1(\infty) = 0$, therefore $\mathcal{G}_1(z)$ is BIBO stable and strictly causal. Then:

$$\begin{aligned} \mathcal{G}(z)^{-1} (\mathcal{G}(z) - \mathcal{G}_1(z)) y(t) &= \mathcal{G}(z)^{-1} \mathcal{F}(z) u(t) + e(t) \\ y(t) - \mathcal{G}(z)^{-1} \mathcal{G}_1(z) y(t) &= \mathcal{G}(z)^{-1} \mathcal{F}(z) u(t) + e(t) \\ y(t) &= \mathcal{G}(z)^{-1} \mathcal{G}_1(z) y(t) + \mathcal{G}(z)^{-1} \mathcal{F}(z) u(t) + e(t) \end{aligned} \tag{3.26}$$

Then it is possible to conclude that

$$\hat{y}(t \mid t-1) = \mathcal{G}(z)^{-1} \mathcal{G}_1(z) y(t) + \mathcal{G}(z)^{-1} \mathcal{F}(z) u(t) + e(t) \tag{3.27}$$

is a one-step-ahead linear predictor of y given the past of y and u.
It can be also proved that $\hat{y}(t \mid t-1)$ is the minimum mean square error predictor.
If at time t only the past data $\{y(s), u(s),\ s < t\}$ are available, a good prediction of $y(t)$ under model $\mathcal{M}(\phi)$ given the past data is

$$\hat{y}_\phi(t \mid t-1) = \mathcal{G}_\phi(z)^{-1} \mathcal{G}_{1,\phi}(z) y(t) + \mathcal{G}_\phi(z)^{-1} \mathcal{F}_\phi(z) u(t) + e(t) \tag{3.28}$$

that yield a prediction error defined as: $\epsilon_\phi(t) \doteq y(t) - \hat{y}_\phi(t \mid t-1)$.
Considering the whole dataset:

$$\begin{aligned} u_{[1,N]} &\doteq [u(1), \ldots, u(N)]^T \\ y_{[1,N]} &\doteq [y(1), \ldots, y(N)]^T \end{aligned} \tag{3.29}$$

it is possible to compute the prediction at every time step and compare it with the actual output. The predictor in (3.28) however must be initialized with initial conditions: since they are unknown as well, they are set ot zero. By doing that it is then possible to compute an approximation of $\epsilon_\phi(t)$ for $t = 1, \ldots, N$.
From the above observation an estimate of ϕ is given by minimizing the term:

$$\frac{1}{\sigma^2} \sum_{t=1}^{N} \epsilon_\phi (t)^2 \tag{3.30}$$

or equivalently

$$V_N (\phi) \doteq \frac{1}{N} \sum_{t=1}^{N} \epsilon_\phi (t)^2 \tag{3.31}$$

since the true parameter σ^2 is unknown, and the expression (3.30) and (3.31) yield the same minimum. Accordingly the *Prediction Error Minimization* (PEM) estimate is:

$$\hat{\phi}_{PEM} \left(y_{[1,N]}, u_{[1,N]} \right) = \text{argmin}_{\phi \in \Phi} V_N (\phi) \tag{3.32}$$

The corresponding estimated model is $\mathcal{M}(\hat{\phi}_{PEM}(y_{[1,N]}, u_{[1,N]}))$.
Moreover an estimate of the noise variance σ^2, and therefore a reasonable value for the parameter φ, is given by:

$$\hat{\sigma}^2 \left(y_{[1,N]}, u_{[1,N]} \right) = \frac{1}{N} \sum_{t=1}^{N} \epsilon_{\hat{\phi}_{PEM}} (t)^2 \tag{3.33}$$

3.5 Kernel-based PEM Method

The kernel-based PEM method is a choice criterium for the best parameter ϕ which is the analogue of the Regularized Least-Squares principle in the static case.
This kind of approach deals with non-parametric models under a Bayesian perspective. For every model in (3.24) it can instead be considered a finite

version in which transfer functions are truncated to their practical length: indeed given that a transfer function $F(z) = \sum_{k=0}^{\infty} f_k z^{-k}$ is BIBO stable, it is known that $f_k \to 0$ as $k \to \infty$. Therefore since both $G(z)^{-1}$ and $G(z)^{-1}F(z)$ are analytic, causal and BIBO stable, any non-parametric model can be approximated through an ARX model given by:

$$A(z)\, y(t) = B(z)\, u(t - n_k) + e(t) \tag{3.34}$$

where $A(z)$ and $B(z)$ are the truncated approximations of $G(z)^{-1}$ and $G(z)^{-1}F(z)$, respectively. Consider the non-parametric Output-Error model:

$$y(t) = F(z)\, u(t - n_k) + e(t) \tag{3.35}$$

where $F(z) = \sum_{k=0}^{\infty} f_k z^{-k}$ is a random transfer function, i.e. f_ks are random variables. Its practical approximation is given by the following ARX model:

$$\mathcal{M}(\phi) \; : \quad y(t) = B(z)\, u(t - n_k) + e(t) \tag{3.36}$$

where $B(z) = \sum_{k=0}^{n_B - 1} b_k z^{-k}$ is a random transfer function, and $n_b - 1$ is the practical length of $F(z)$.
Equation (3.36) can be rewritten as

$$y(t) = \varphi(t)^T \phi + e(t) \tag{3.37}$$

where $\varphi(t) = [u(t-1), u(t-2), \ldots, u(t-n_B)]^T$, $\phi = [b_0, b_1, \ldots, b_{n_B - 1}]^T$ is a random vector such that $\mathbb{E}[\phi] = 0$ and $\mathbb{E}[\phi\phi^T] = K = K^T \succ 0$. K is called *kernel matrix*. ϕ contains the coefficients of the impulse response of $B(z)$.
From $\mathbb{E}[\phi\phi^T] = K$ it can be derived $\mathbb{E}[\|\phi\|^2_{k^{-1}}] = n_B$. The latter means that $\|\phi\|^2_{k^{-1}}$ is bounded in mean and so it is reasonable to assume that a realization is bounded too. Given data (y^N, u^N), drawing inspiration from the PEM method from the previous section and in view of the last observation, an estimate of ϕ is given by minimizing the fit term $\frac{1}{\sigma^2} \sum_{t=1}^{N} \epsilon_\phi(t)^2$ under the constraint that $\|\phi\|^2_{k^{-1}}$ is bounded.

It can be proved that the problem is equivalent to:

$$\begin{aligned}\hat{\phi}_K\left(y^N, u^N\right) &= \operatorname{argmin}_{\phi\in\Phi} \frac{1}{\hat{\sigma}^2}\sum_{t=1}^{N} \epsilon_\phi\left(t\right)^2 + \|\phi\|^2_{K^{-1}} \\ &= \operatorname{argmin}_{\phi\in\Phi} \sum_{t=1}^{N} \epsilon_\phi\left(t\right)^2 + \hat{\sigma}^2\|\phi\|^2_{K^{-1}}\end{aligned} \tag{3.38}$$

that is the *kernel-based PEM* estimate using kernel K.
The corresponding model is $\mathcal{M}(\hat{\phi}_K(y^N, u^N))$.
Different kernels can be designed according to the prior knowledge that is available for the estimation problem at hand.
Consider the ARX model

$$y\left(t\right) = B\left(z\right)u\left(t - n_k\right) + e\left(t\right) \tag{3.39}$$

The *Diagonal* kernel is represented by a matrix of the form

$$K = \lambda\operatorname{diag}\left(\beta, \beta^2, \ldots, \beta^{n_B}\right) \tag{3.40}$$

with $0 < \beta < 1$, and it is possible to prove that the use of this kernel imposes BIBO stability on $B(z)$ as $n_B \to \infty$. In this case there are two hyper-parameters, therefore $\varphi = [\lambda, \beta]$.
In order to further impose the boundedness of the first derivative for thee impulse response of $B(z)$ while keeping the constraint on BIBO stability it is possible to consider the *Tuned-Correlated* kernel that is given by

$$K = \lambda K_{TC} \qquad \text{where} \quad \left(K_{TC}\right)_{i,j} = \min\left\{\beta^i, \beta^j\right\} \tag{3.41}$$

with the same hyper-parameters as before.
In both kernels $\sqrt{\beta}$ can be understood as an upper bound of the slowest mode in $B(z)$.

4
Current Approaches

In this chapter the state of the art in Reinforcement Learning is briefly summarized, with the emphasis on the dichotomy between *model-based* perspective and *model-free* methods. First of all the main framework is restated in a control-oriented fashion, then the two main approaches will be presented with the help of actual algorithms that follow one or the other methodology.

4.1 Control with Unknown Dynamics

Both Control Theory and Reinforcement Learning study how to use past data to enhance the future manipulation of a dynamical system, although from different viewpoints: the latter is a data driven approach while the former heavily relies on a model of the system. Moreover, while in control engineering the main focus may be to design a controller that drives the system as specified by a reference signal with a robust solution, Reinforcement Learning algorithms achieve impressive performances but they are neither safe nor reliable. Indeed in every discipline that is strongly involved with any kind of learning, there is always the possibility that the algorithm is missing a crucial aspect of the environment that had never manifested it-

self before, and thus a slight change in the setting might result in very poor performances or even dangerous behaviour.
Since these two fields share the same aim to design systems that use richly structured perception, perform planning and control that adequately adapt to environmental changes, and exploit safeguards when surprised by a new scenario, it could be useful to consider both perspective and how they address the same problem. Indeed, understanding how to properly analyse, predict and certify such systems require insight from current machine learning practice and from the applied mathematics of optimization, statistics and control theory.
The same Reinforcement Learning framework presented in Chapter 2, Section 2.5 can be restated with the shape of a classical optimal control problem.
The system is represented by a dynamical system governed by the difference equation:

$$x_{t+1} = f_t\left(x_t, u_t, e_t\right) \tag{4.1}$$

whit x_t as the state of the system, u_t as the control action, and e_t as a random disturbance. f_t is the rule that maps the current state, control action, and disturbance to the next state for the agent. The cost is a function of the state and the input, $C(x_t, u_t)$, that is generated by the system and it is sent to the agent.
As it has been previously illustrated, the goal of the learner is to minimize the cost. This formally corresponds to solving the problem:

$$\begin{aligned} \text{minimize} \quad & \mathbb{E}_{e_t}\left[\sum_{t=t_0}^{T} C_t\left(x_t, u_t\right)\right] \\ \text{subject to} \quad & x_{t+1} = f_t\left(x_t, u_t, e_t\right) \end{aligned} \tag{4.2}$$

That is, the aim is to minimize the expected reward over T time steps with respect to the control sequence u_t, subject to the dynamics specified by the state-transition rule f_t.
It is assumed that u_t is to be chosen having the knowledge of states x_{t_0} through x_t and previous input u_{t_0} through u_{t-1}.
The term *trajectory* will be referred to as a sequence of states and control

actions generated by a dynamical system, defined as:

$$\tau_t \doteq (u_{t_0}, \dots, u_{t-1}, x_{t_0}, \dots, x_t) \tag{4.3}$$

Note that, being the cost a function of the state and input at a certain time, the only way in which the agent can affect it is through the control input, that is in fact the only free variable.
Since the environment is uncertain, the dynamics f_t are stochastic, and the agent should be able to observe the state before deciding upon the next action, in order to mitigate the unreliability of the model. In a more general framework, the agent has access to the whole trajectory, to improve its decision-making. Therefore, rather than optimizing over deterministic sequences of controls u_t, the minimization is carried out over policies.
A control policy π is a function that takes a trajectory from a dynamical system and outputs a new control action. Hence, in the statement (4.2), the input control will take the form of $u_t = \pi_t(\tau_t)$.
Policies π_t are the only decision variables of the problem.
As it has already been highlighted, the challenging part comes from the uncertainty about the model. This is often the case, and it could be the result of a system that is too difficult to model, or it could refer to an actual lack of knowledge of inner dynamics. In any case, it is impossible to solve Problem (4.2) using standard optimization methods: it is first necessary to learn something about the system, and then exploit this knowledge in order to choose the best policy.
The main paradigm in Reinforcement Learning suggest to decide on a policy π and a time horizon L; then the policy is implemented on the real system and it returns a trajectory τ_L and a sequence of costs $\{C_t(x_t, u_t)\}$ for $t = t_0, \dots, L$. The aim is to find the policy that minimizes the cost with the fewest total number of samples computed by the oracle, that is, by questioning the system. By running m queries with horizon length L, the total cost, in terms of samples, would be mL. This is the *oracle model* and the procedure falls within the context of the *episodic* Reinforcement Learning.
The trade-off between exploration and exploitation that has been introduced in Chapter 2, Section 2.4 becomes now clearer: the aim is to obtain an high

expected reward for the derived policy, while keeping the number of oracle queries as small as possible.
Each episode returns a complex feedback signal of states and costs: this kind of oracle it is considerably more complicated than the ones typically considered in oracle models for optimization.
The problem of finding the best policy is still ill-posed, since there is no clear definition of the optimality measure that should be satisfied. It is possible to evaluate the cost given a fixed number of samples, or evaluating the number of samples necessary to reach a certain threshold for the cost: one algorithm could be better than another in one case and worse in the other.

4.2 State of the Art

Nowadays two different viewpoints have become the cornerstones in dealing with the state-transition function uncertainty, differing from each other for the way in which they exploit the interaction with the real system: *Model-Based* and *Model-Free* Reinforcement Learning.
The model-based approach aims at learning a model of the system dynamics, fitting it to previously observed data, and then use strategies from optimal control theory to solve Problem (4.2).
In model-free Reinforcement Learning the model is neglected, eschewing the need for learning the state-transition function, and it is directly sought a map from observation to actions.
The latter methodology aims to solve optimal control problem only by probing the system and improving strategies based on past rewards and states. These are in some sense more direct methods, since they deal with the true system itself, trying to reach the goal. Indeed the knowledge of the model is not required for learning the task and often complex details are involved in order to simulate a dynamical system. Moreover, different models of the system can lead to the same policy, therefore retrieving a rich and detailed model might not be a good approach, if it does not improve the resulting policy.

4.3 Model-Based Reinforcement Learning

The main obstacle in dealing with the problem is represented by the uncertainty in the state-transition function, therefore the most obvious strategy would be to come up with an estimate of the system dynamics and to rely on this predictive model to find a solution for the prescribed control problem.
The estimate of the state-transition function is called *nominal model*, and the input control designed assuming the estimated model is true is referred to as *nominal control*.
System Identification is the branch of Automation Engineering that aims at estimating models for dynamical systems from data, addressing the problem posed by the lack of knowledge of the state-transition function. It is a wide field, that differs from conventional estimation in many aspects. Distinctive features are the need for a careful design of the training inputs, in order to excite various degrees of freedom and at the same time avoid non-linear phenomena; and correlation over time of the dynamical outputs with the parameters to be estimated, the inputs fed to the system and the stochastic disturbances. A further insight in System Identification can be found in Chapter 3, while in the following is considered a very simple strategy just to set up the model-based framework.
All is required is a predictor of x_{t+1} built from the trajectory history. The simplest approach would be to inject a random probing sequence u_t for control and then measure how the state responds. Up to stochastic noise, the sought function should satisfy:

$$x_{t+1} \approx g\left(x_t, u_t\right) \tag{4.4}$$

where g is some model aiming to approximate the true dynamics.
Clearly g might be a parametric function based on first-principles physical model or it may arise from a neural network as a non-parametric approximation. This generic state-transition function can then be made suitable through supervised learning using collected data. The trained function will be denoted as $\hat{g}$.
Clearly the predictor will not match exactly the true dynamical system,

therefore a random variable e_t is taken into account as a model for the noise process.
Relying on the estimated function, it is possible to solve the optimal control problem:

$$\begin{aligned} \text{minimize} \quad & \mathbb{E}_{e_t}\left[\sum_{t=t_0}^{T} C_t\left(x_t, u_t\right)\right] \\ \text{subject to} \quad & x_{t+1} = \hat{\varphi}\left(x_t, u_t\right) + w_t,\ u_t = \pi\left(\tau_t\right) \end{aligned} \tag{4.5}$$

However the solution will not be an optimal control policy π_t^*, as 4.5 is not the initial problem that had to be addressed: the model is incorrect, and moreover this formulation requires some plausible model of the noise process.
The main drawback of this approach is indeed *model bias*: the algorithm needs to rely on the estimated model in order to search for the best policy, thus claiming the correctness of the estimated state-transition function and neglecting possible approximations or errors introduced by the system identification procedure.
Nonetheless, if $\hat{g}$ and f are close, this approach might work well in practice.

4.3.1 PILCO

In recent literature, one of the most effective algorithms that works under the model-based perspective and still addresses the problem of model bias, is PILCO.
The main idea in the work proposed by Deisenroth & Rasmussen [2] is to learn a probabilistic dynamics model and explicitly incorporate model uncertainty into long-term planning. Instead of relying on the estimated model, a posterior distribution over transition functions is considered, allowing to reduce model bias and to quantify the level of uncertainty about the model.
By maintaining the framework presented above, with the state space being $\mathbb{R}^D$ and the input space $\mathbb{R}^F$, the aim of the algorithm is to minimize the

expected return averaged on the initial state, over policies $\pi(\tau_t, \theta)$:

$$J_\pi(\theta) = \sum_{t=t_0}^{T} \mathbb{E}_{x_t}\left[C(x_t, u_t)\right], \quad x_0 \sim \mathcal{N}(\mu_0, \Sigma_0) \tag{4.6}$$

The probabilistic dynamics model is implemented as a Gaussian Process, where tuples $(x_{t-1}, u_{t-1}) \in \mathbb{R}^{D+F}$ are used as training inputs and $\Delta_t = x_t - x_{t-1} + \epsilon \in \mathbb{R}^D$, $\epsilon \sim \mathcal{N}(0, \Sigma_\epsilon)$, $\Sigma_\epsilon = \mathrm{diag}([\sigma_{\epsilon 1}, \ldots, \sigma_{\epsilon D}])$ as training targets.
The Gaussian Process yields one-step predictions

$$\begin{aligned} p(x_t \,|\, x_{t-1}, u_{t-1}) &= \mathcal{N}(x_t \,|\mu_t, \Sigma_t) \\ \mu_t &= x_{t-1} + \mathbb{E}_f[\Delta_t] \\ \Sigma_t &= \mathrm{var}_f[\Delta_t] \end{aligned} \tag{4.7}$$

Since minimizing and evaluating J_π in Eq. (4.6) requires long-term prediction of the state evolution, the state distributions $p(x_1), \ldots, p(x_T)$ are obtained by cascading one-step predictions in Eqs. (4.7). This requires mapping uncertain test inputs through the Gaussian Process dynamics model. For predicting x_t from $p(x_{t-1})$, it is required the joint distribution $p(\tilde{x}_{t-1}) = p(x_{t-1}, u_{t-1})$. Taking the control input as a function of the state alone and the parameter, i.e. $u_{t-1} = \pi(x_{t-1}, \theta)$, the necessary joint distribution is computed by approximating the input as a Gaussian random variable, and again considering the Gaussian approximation of the desired joint distribution.
Assuming then the knowledge of $p(\tilde{x}_{t-1}) = \mathcal{N}(\tilde{x}_{t-1} \,|\, \tilde{\mu}_{t-1}, \tilde{\Sigma}_{t-1})$, it is possible to predict the distribution:

$$p(\Delta_t) = \int p(f(\tilde{x}_{t-1}) \,|\, \tilde{x}_{t-1})\, p(\tilde{x}_{t-1})\, \mathrm{d}\tilde{x}_{t-1} \tag{4.8}$$

where the transition probability $p(f(\tilde{x}_{t-1}) \,|\, \tilde{x}_{t-1})$ is obtained from the Gaussian Process distribution.
Finally, a Gaussian approximation to the distribution $p(x_t)$ is given as $\mathcal{N}(x_t \,|\, \mu_t, \Sigma_t)$

with

$$\begin{aligned} \mu_t &= \mu_{t-1} + \mu_\Delta \\ \Sigma_t &= \Sigma_{t-1} + \Sigma_\Delta + \mathrm{cov}[x_{t-1}, \Delta_t] + \mathrm{cov}[\Delta_t, x_{t-1}] \\ \mathrm{cov}[x_{t-1}, \Delta_t] &= \mathrm{cov}[x_{t-1}, u_{t-1}]\Sigma_u^{-1}\mathrm{cov}[u_{t-1}, \Delta_t] \end{aligned} \tag{4.9}$$

Both μ_t and Σ_t are functionally dependent on the mean μ_u and the covariance Σ_u of the control signal, and so on θ, through $\tilde{\mu}_{t-1}$ and $\tilde{\Sigma}_{t-1}$. Hence, it is possible to analytically compute the gradient of the expected return J_π with respect to the policy parameter θ, and thus update its value following the gradient descent direction in order to get every step closer to the minimum of J_π.
The analytic gradient computation of J_π should be much more efficient than estimating policy gradient through sampling.
The overall procedure is sketched in Algorithm 1.

Algorithm 1 PILCO

1: **init:** Sample controller parameters $\theta \sim \mathcal{N}(0, I)$. Apply random control signals and record data;
2: **repeat**
3: Learn probabilistic dynamics model, using all data;
4: Model-based policy search;
5: **repeat**
6: Approximate inference for policy evaluation: get $J_\pi(\theta)$;
7: Gradient-based policy improvement: get $\mathrm{d}J_\pi(\theta)/\mathrm{d}\theta$;
8: Update parameters θ;
9: **until** convergence;
10: **return** θ^*;
11: Set $\pi^* \leftarrow \pi(\theta^*)$;
12: Apply π^* to system and record data;
13: **until** task learned.

4.4 Model-Free Reinforcement Learning

Every algorithm that does not search for a model of the environment but addresses the problem of minimizing the cost by taking actions falls under

this category.
This paradigm is focused on directly learning a policy function from episodic experiences, ignoring the state-transition function and relying only on rewards as outcomes of a specific action taken into a specific state. These policy driven methods turn the problem of Reinforcement Learning into derivative-free optimization.
First of all it is reviewed a general paradigm for exploiting random sampling in order to solve optimization problem.
Consider in this regard the general unconstrained optimization problem:

$$\text{maximize}_{z \in \mathbb{R}^d} \quad R(z) \tag{4.10}$$

This can also be rewritten as an optimization problem over probability distributions on z:

$$\text{maximize}_{p(z)} \quad \mathbb{E}_p [R(z)] \tag{4.11}$$

Indeed if z^* is the optimal solution to (4.10), then (4.11) will return the same value when placing a δ-function around z^*.
Moreover, being p a probability distribution, it is obvious that the expected value of the reward function can never be higher than the maximal reward achievable by a fixed z. Therefore it it possible to optimize over z or similarly over distribution functions over z.
Of course an optimization over the space of all probability densities would be ill-posed and therefore intractable, hence it is necessary to restrict the class of function over which the optimization is performed.
Considering a family of densities parametrized by a parameter vector θ, $p(z;\theta)$, the problem becomes:

$$\text{maximize}_{\theta} \quad \mathbb{E}_{p(z;\theta)} [R(z)] \tag{4.12}$$

As already noticed, if the family of distribution does not contain all the δ-functions, the resulting optimization problem only provides a lower bound on the optimal value, no matter how good is the distribution that has been found.
However, this representation provides a powerful ad general algorithmic

framework for optimization. Indeed by defining $J(\theta) \doteq \mathbb{E}_{p(z;\theta)}[R(z)]$ it is possible to compute its derivative by using the so called "log-likelihood trick":

$$\begin{aligned}\nabla_\theta J(\theta) &= \int R(z)\,\nabla_\theta p(z;\theta)\,\mathrm{d}z \\ &= \int R(z)\left(\frac{\nabla_\theta p(z;\theta)}{p(z;\theta)}\right) p(z;\theta)\,\mathrm{d}z \\ &= \int \left(R(z)\,\nabla_\theta \log p(z;\theta)\right) p(z;\theta)\,\mathrm{d}z \\ &= \mathbb{E}_{p(z;\theta)}\left[R(z)\,\nabla_\theta \log p(z;\theta)\right]\end{aligned} \tag{4.13}$$

This derivation reveals that the gradient of J with respect to ϑ is the expected value of the function:

$$G(z;\theta) = R(z)\,\nabla_\theta \log p(z;\theta) \tag{4.14}$$

Therefore, by sampling z from the distribution $p(z;\vartheta)$, it is possible to compute $G(z,\vartheta)$ thus retrieving an unbiased estimate of the gradient of J. A common strategy to solve an optimization problem is provided by *Stochastic Gradient Descent*, that is exactly given by updating the parameter following the direction computed above.
This method is very simple to implement: the pseudocode is given in Algorithm 2.

Algorithm 2 REINFORCE

1: **Hyperparameters:** Step-sizes $\alpha_j > 0$.
2: **init:** θ_0 and $k = 0$;
3: **while** ending condition not satisfied **do**
4: Sample $z_k \sim p(z;\theta_k)$;
5: Set $\theta_{k+1} = \theta_k + \alpha_k R(z_k)\nabla_\theta \log p(z_k;\theta_k)$
6: Set $k \leftarrow k+1$;
7: **end**

The benefits of this approach are the ease of implementation and as long as it is possible to efficiently sample from $p(z;\theta)$, this algorithm can tackle any problem.
However, this method operates on stochastic gradients of the sampling dis-

tribution, while the quantity to be maximized, $R(z)$ can be only accessed through function evaluations. Direct search approaches using log-likelihood trick are necessary derivative free optimization methods and therefore are less effective than methods that compute actual gradients, especially when the function evaluations are noisy. Moreover, the choice of distribution can lead to very high variance in stochastic gradients, resulting in the need for many samples to be drown in order to find a stationary point.

4.4.1 Policy Gradient

The optimal policy for Problem (4.2) is always deterministic, nonetheless the main idea behind policy gradient is to use probabilistic policies. These are optimal for other optimization problems such as control of partially observed Markov decision processes or zero-sum games.
By considering a parametric and randomized policy, the control input u_t will be sampled from a distribution $p(u \mid \tau_t, \theta)$ that is a function of the currently observed trajectory and the parameter vector θ.
A probabilistic policy induces a probability distribution over trajectories:

$$p(\tau; \theta) = \prod_{t=t_0}^{T-1} p(x_{t+1} \mid x_t, u_t)\, p(u_t \mid \tau_t; \theta) \tag{4.15}$$

Moreover, defining the reward of a trajectory as:

$$R(\tau) = \sum_{t=t_0}^{T} R_t(x_t, u_t) \tag{4.16}$$

the optimization problem for Reinforcement Learning takes in fact the form of Problem (4.12).
Policy Gradient thus proceeds by sampling a trajectory using the probabilistic policy with parameters θ_k, and then updating using Algorithm 2.
Following this approach, it should be remembered that the gradient of J with respect to θ, computed with the log-likelihood trick, is not an explicit function of the underlying dynamics: by shifting to distribution over poli-

cies, the burden of optimization is left to the sampling procedure.

5
A Mixed Approach

This Chapter is focused on illustrating the main contribution of this work, that is a mixed approach that exploit ideas from the two well-established paradigms analysed before.
As it has been explained in Chapter 4, both model-based and model-free Reinforcement Learning perspectives have their merits and defects, making them suitable for different scenarios.
Model-based Reinforcement Learning can be seen as a standard approach in Control Engineering, since the main framework is based on a classical optimal control problem, with the difference that the state-transition function is at least partially unknown, and should be estimated through System Identification. Once the data-driven model has been built, the best policy can be found by resorting to optimal control techniques. It is then possible to further improve the retrieved policy by relearning the model every time as in PILCO (see Chapter 4, Section 4.3.1), that however significantly slows down the algorithm.
Model-free methods seem entirely different approaches, completely neglecting the role of the model and considering instead only the map from pair state-action to reward. Without looking for a whole knowledge of the system it is dealing with, this kind of perspective is much more committed

to the idea of trying a new input, though based on previous history, and observing if the reward improves or not, leading to a less safe search paradigm. Indeed the selection of the policy for the very first trials on the system must be almost completely random, since there is no information available on the system. This can lead to the choice of a policy that jeopardises the proper functioning of a mechanical system, or any other system that has precise working regions.
Although it is true that the same problem affects the System Identification procedure that needs to be carried out following the model-based perspective, in this latter case it is considered a well-established framework where it is possible to proceed with caution. Therefore model-based approaches guarantee a safer interaction with the unknown system, though leading to the need for learning a detailed model of the environment, which can be an unnecessary step. Indeed sometimes only some characteristic features of the model are useful to point towards a good policy: in such a scenario a very detailed model and a simple one may carry the same amount of useful information, resulting in a waste of data in the first case. Surely having a straight interaction with the system through a certain policy is a much more direct way of querying the system at hand, and return a more reliable point estimate of the overall cost.
The idea from which the main contribution of this Master Thesis was originated is to mix the two approaches in order to exploit the benefits of both of them, while attenuating their drawbacks. In particular the core thought is to run the model-based procedure with only a few samples in order to retrieve a raw model of the dynamical system and to use this knowledge as a regularizer. As second step, following the model-free perspective, some policies are tried over the real system generating punctual estimates of the cost, in order to provide samples that should be fitted by the reconstructed function.
This method therefore provides a way to merge the two class of Reinforcement Learning algorithms in a whole paradigm, that should be more flexible than following just one of the single approaches.

5.1 Formalization

In order to better understand how the aforementioned idea can be stated in a more formal framework, the main elements of a Reinforcement Learning paradigm are placed in a mathematical contest.
As far as the system to be addressed is concerned, the following model is considered:

$$\begin{cases} x_{t+1} = f(x_t, u_t) \\ y_t = h(x_t) + e_t \end{cases} \tag{5.1}$$

where f is the unknown state-transition function representing the environment, x_t is the state in which the agent is located at time t, u_t is the control input, y_t is the output of the system, e_t represents the noise disturbance at the output channel and h is the output function of the system, which models the fact that the whole state variable may be not accessible and so the only information about the state of the system is the measurable output. This choice has been made for the sake of clarity, however the output function can be merged together with the state-transition function as $y_t = h \circ f(u_t) + e_t$ so everything can be thought in terms of state, which becomes equivalent to the output.
As it is well known from Chapter 2 the goal of a Reinforcement Learning algorithm is set by defining the cost function $c(y_t)$ that here depends on the observable variables. By fixing a time interval T the return J is given by:

$$J(t_0) = \sum_{t=t_0}^{t_0+T-1} \lambda^{t-t_0} c(y_t) \tag{5.2}$$

where λ is the discount factor.
However, as it has been explained in the refined framework for control (see Chapter 4, Section 4.1), the policy is the real decision variable of the problem, providing the input that excites the system as a function of the past trajectory. Dealing with a system of the form in (5.1), the only available quantities are the outputs, which can be measured and thus be collected.

Therefore a more suitable version of the concept of trajectory is given by:

$$\varphi_t = [y_{t-1}, \dots, y_{t-T}, u_{t-1}, \dots, u_{t-T}, u_{t-T-1}] \tag{5.3}$$

that is related to the past behaviour of the system, being the state-transition function unknown thus making impossible to formulate hypotheses on future states. y_{t-T} represent the initial condition of the system, before any input is applied.
In order to set up a search for the best policy with the goal of minimizing the return, it is necessary to define a class to look in, otherwise the problem is ill-posed: in the following it will be considered a general parametrization of policies depending on the trajectory and the current output $\pi(\varphi_t, \theta, y_t)$, with θ being a parameter vector belonging to $\Theta \subset \mathbb{R}^d$.
The input at time t will then be given by $u_t = \pi(\varphi_t, \theta, y_t)$.
From now on, following the model-based approach, the state-transition function should be reconstructed and then used to find the best value for θ relying on the model retrieved. This can be done by using $N_{mb} \in \mathbb{N}$ input-output pairs in order to run the estimation procedure, that are the only envisaged interaction with the unknown environment.
The return, as a function of the parameter, will be then given by:

$$\begin{aligned} J(t_0, \theta) &= \sum_{t=t_0}^{t_0+T} \lambda^{t-t_0} c(y_t) \\ \text{subject to} \quad &\begin{cases} x_{t+1} = \hat{f}(x_t, \pi(\varphi_t, \theta, y_t)) \\ y_t = h(x_t) \end{cases} \end{aligned} \tag{5.4}$$

where $\hat{f}$ is the learned state-transition function. In this way the return can be analytically computed for any θ.
The best policy is then simply given by $\pi(\varphi_t, \theta^*, y_t)$, where θ^* is the minimum point of the return:

$$\theta^* = \text{argmin}_{\theta \in \Theta} J(t_0, \theta) \tag{5.5}$$

On the other hand, model-free methods suggest to fix a certain $\theta = \theta_k$, apply the corresponding control, and then to perturb θ along the Stochastic

Gradient Descent direction computed with the support of collected data. In this case is not important to have a fully featured function for the return, nonetheless a reconstruction based on point measures will be given by:

$$\hat{J}\left(t_0, \theta\right) = \operatorname{argmin}_{J \in \mathcal{J}} \frac{1}{N_{tr}} \sum_{k=1}^{N_{tr}} \left\| J\left(t_0, \theta\right) - J\left(t_0, \theta_k\right) \right\|^2 \tag{5.6}$$

for example relying on a least squares estimate of the return function. $\mathcal{J}$ is a space of functions that is needed to keep the problem well-posed, from which it should be selected the function that best fit the collected data. These are given by $J(t_0, \theta_k)$, for $k = 1, \ldots, N_{tr}$, which represent actual values of the return for the corresponding values of θ.
Clearly in order to get a single reliable point estimate of the return function there is the need to interact with the system more than once: indeed every value is obtained by summing T terms for the cost. Therefore system queries are carried out at a higher cost, since the total number of interactions with the system is $N_{mf} = N_{tr}T \in \mathbb{N}$.
The proposed method rely on both model-based and model-free perspective: instead of following a single procedure, these two approaches can be mixed together in a regularized framework. The reconstruction carried out following model-free methods can indeed be seen as a fitting term, since this approach provide actual samples from the return; while the procedure of model-based Reinforcement Learning can supply some knowledge about the state-transition function and so about the shape of the return, in order to improve the generalization capabilities of the estimated return function, thus allowing to choose a better θ in term of rewards. In fact, following the idea suggested in PILCO, by considering a probability distribution of state-transition function instead of relying just on the simple estimate, it is possible to avoid model bias and to build a regularization term that prefers more plausible models. This however leads to a slight change in the model-based paradigm.
The overall reconstruction of the return as a function of the parameter fol-

lowing the combined approach leads to:

$$\hat{J}\left(t_0,\theta\right)=\text{argmin}_{J\in\mathcal{J}}\left\{\frac{1}{N_{tr}}\sum_{k=1}^{N_{tr}}\left\|J\left(t_0,\theta\right)-J\left(t_0,\theta_k\right)\right\|^2+\rho\left\|J\right\|_{\mathcal{H}}^2\right\} \tag{5.7}$$

where $\mathcal{H}$ is a suitable space of functions inducting a particular norm based on the model dynamics, and ρ is an hyper-parameter that tunes the weight between the two terms.
In standard regularization the penalty term has the effect of shifting the minimum point towards a simpler model, that explains data sufficiently well. Indeed if the function to be retrieved can take on an arbitrary degree of complexity, and if solely the fitting term is considered for the reconstruction, then the resulting function would interpolate all data fitting also noise and ending uo having poor generalization capabilities and instability of the solution to sample perturbations.
Nonetheless, by weighting models with the aim of expressing a preference over more stable functions it is possible to balance the effect of data fitting and to induce the reconstruction to perform better over fresh samples of the sought distribution.
However in Problem (5.7) the role of the penalty term is different: it should reflect a preference for models that are contained in a specific class, built according to the knowledge gained from the identification procedure applied to the system. In order to follow such an approach, it would be necessary to dispose of a distribution of J over θ, estimated by data at hand.
Suppose for the moment that the estimation procedure for the state-transition function returned a probability distribution of J, describing how much variability the model can have based on data collected. Than the norm induced by the space $\mathcal{H}$ should have a value inversely proportional to the probability of the model, which indeed reflect a preference over a more plausible model. That is, a model fitting data coming from the model-free approach perfectly but having a very low probability in the distribution retrieved by the system identification procedure will be discarded in favour of a model fitting data slightly worse but having a higher probability of being the true model.
Therefore the main issue now is how to build such a functional space, retriev-

ing a probability distribution over return functions according to the measured density over state-transition functions, thanks to the System Identification procedure.
In what follows a very simple framework will be considered, in order to better understand how this procedure might work and to introduce a theoretical approach for the solution of the problem above.

5.2 A First Theoretical perspective

This first attempt is based on propagating the distribution resulting from the system identification procedure thus obtaining a distribution over future states and future rewards, which should allow to compute an approximation for the return density.
To keep this scenario as simple as possible, consider a Finite Impulse Response filter for the system identification procedure, given by:

$$y_t = [h * u]_{t-1} + e_t \tag{5.8}$$

that can be seen as a simplified version of the Output-Error model with $F(q) = 1$, where $e_t \sim \mathcal{N}(0, \sigma^2)$ is the model for the noise at the output channel and $h \in \mathbb{R}^n$ is the FIR filter with practical length equal to n.
Suppose to excite the system with the input vector $u = [u_1, \dots, u_{N_{mb}}]$ yielding the following measurements:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N_{mb}} \end{bmatrix} = \Phi \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{N_{mb}} \end{bmatrix}, \quad \Phi = \begin{bmatrix} u_0 & 0 & \dots & 0 \\ u_1 & u_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ u_{N_{mb}-1} & u_{N_{mb}-2} & \dots & u_{N_{mb}-n} \end{bmatrix} \tag{5.9}$$

where Φ is a Toeplitz matrix belonging to $\mathbb{R}^{N_{mb} \times n}$, considering the input preceding 0 as zeroes. u_0 is the input generated by the initial condition y_0.
By running a kernel-based Prediction Error Minimization process with Tuned-Correlated kernel as regularizer, it is possible to obtain the distribution over

models as:

$$h \sim \mathcal{N}\left(\hat{h}, \Sigma\right) \tag{5.10}$$

In order to keep the framework as simple as possible, a quadratic cost is considered, expressing the goal of following a given reference r_t for the system:

$$c\left(y_t\right) = \left(y_t - r_t\right)^2 \tag{5.11}$$

The control law is governed by the policy, that depends on the previous trajectory, the parameter θ and the current output y_t:

$$u_t = \pi\left(\phi_t, \theta, y_t\right) \tag{5.12}$$

Finally, by taking the discount factor as $\lambda = 1$ the return to be minimized is given by:

$$J\left(\theta, \varphi_t^+\right) = \sum_{t=t_0}^{t_0+T-1} \left(y_t - r_t\right)^2 \tag{5.13}$$

where φ_t^+ is the future trajectory of the system.
Recall that the main purpose is to find a distribution over the return function for a fixed value of $\theta = \bar{\theta}$, exploiting the knowledge of past trajectory and the distribution over h, i.e.:

$$J\left(\bar{\theta}, \varphi_t^+ \mid \varphi_t\right) \sim \textbf{?} \tag{5.14}$$

By propagating the probabilistic dynamics of the model, it is possible to retrieve a probabilistic description of the future outputs $y_{t+1}, \ldots, y_{t+T-1}$ from which it can be derived an accurate characterization of the final return function.
Following this perspective the expectation and the variance of y_t given φ_t are respectively denoted as $\hat{y}_t$ and Σ_t^y, and relying on a Gaussian approximation of the output variable it is possible to write:

$$y_t \mid \varphi_t \sim \mathcal{N}\left(\hat{y}_t, \Sigma_t^y\right) \tag{5.15}$$

An elementary computation then yields:

$$\hat{y}_t = \mathbb{E}\left[[h * u]_{t-1} + e_t\right] = \mathbb{E}\left[u^T_{[t-1,t-n]}h\right] = u^T_{[t-1,t-n]}\hat{h} \tag{5.16}$$

since $u_{[t-1,t-n]}$ is completely determined from φ_t.

$$\begin{aligned}\Sigma^y_t = \mathrm{Var}\left[[h * u]_{t-1} + e_t\right] &= \mathrm{Var}\left[\left[u^T_{[t-1,t-n]}h\right]\right] + \mathrm{Var}\left[e_t\right] \\ &= u^T_{[t-1,t-n]}\Sigma u_{[t-1,t-n]} + \sigma^2\end{aligned} \tag{5.17}$$

The key fact that make the derivation of these quantity so easy is that the past input is known and thus the variability on y_{t+1} is due only to the probability distribution of h.
The computations in fact become more cumbersome as the time index progresses:

$$\begin{aligned}y_{t+1} \,|\, \varphi_t = [h * u]_t + e_{t+1} &= u^T_{[t,t-n-1]}h + e_{t+1} \\ &= u_t h_1 + u^T_{[t-1,t-n-1]}h_{[2,n]} + e_{t+1}\end{aligned} \tag{5.18}$$

that is not easy to be handled, since

$$u_t = \pi\left(\varphi_t, \bar{\theta}, (y_t \,|\, \varphi_t)\right) \tag{5.19}$$

where $y_t \,|\, \varphi_t$ is a random variable.
As it can be seen from (5.19), the uncertainty on the output variable y_t makes the input at the same time u_t a random variable as well. Therefore, without a meaningful description of the distribution of the input variable, it is impossible to advance further in computing $y_{t+1}|\varphi_t$. By assuming again a Gaussian distribution for u_t, is then necessary to determine $\mathbb{E}[u_t] \doteq \hat{u}_t$ and $Var[u_t] \doteq \Sigma^u_t$.
This can be done by resorting to a Taylor expansion of the first order, which considerably simplifies the problem at the price of introducing another significant approximation in the framework:

$$\begin{aligned}u_t &= \pi\left(\varphi_t, \bar{\theta}, y_t\right) \\ &\approx \pi\left(\varphi_t, \bar{\theta}, y_t\right)\big|_{y_t=\hat{y}_t} + \frac{\partial \pi}{\partial y_t}\left(\varphi_t, \bar{\theta}, y_t\right)\Big|_{y_t=\hat{y}_t}(y_t - \hat{y}_t)\end{aligned} \tag{5.20}$$

thus retrieving a complete yet approximate description of the density of u_t as

$$u_t \sim \mathcal{N}\left(\pi\left(\varphi_t, \bar{\theta}, \hat{y}_t\right), \Pi_t^T \Sigma_t^y \Pi_t\right) \tag{5.21}$$

where

$$\Pi_t = \frac{\partial \pi}{\partial y_t}\left(\varphi_t, \bar{\theta}, y_t\right)\Big|_{y_t=\hat{y}_t} \tag{5.22}$$

It is now possible to proceed with the computation related to $y_{t+1}|\varphi_t$ that yield, for the expectation:

$$\begin{aligned}\mathbb{E}\left[y_{t+1} \,|\, \varphi_t\right] &= \mathbb{E}\left[u_t h_1 + u_{[t-1,t-n-1]}^T h_{[2,n]} + e_{t+1}\right] \\ &= \mathbb{E}\left[u_t h_1\right] + \mathbb{E}\left[u_{[t-1,t-n-1]}^T h_{[2,n]}\right] + \mathbb{E}\left[e_{t+1}\right] \\ &= \mathbb{E}\left[u_t\right]\mathbb{E}\left[h_1\right] + u_{[t-1,t-n-1]}^T \hat{h}_{[2,n]}\end{aligned} \tag{5.23}$$

where in the last step it has been assumed that u_t and h_1 were uncorrelated, i.e. $\operatorname{cov}(u_t, h_1) = 0$. The variance is approximated as:

$$\begin{aligned}\operatorname{Var}\left[y_{t+1} \,|\, \varphi_t\right] &= \operatorname{Var}\left[u_t h_1 + u_{[t-1,t-n-1]}^T h_{[2,n]} + e_{t+1}\right] \\ &= \operatorname{Var}\left[u_t h_1\right] + \operatorname{Var}\left[u_{[t-1,t-n-1]}^T h_{[2,n]}\right] + \operatorname{Var}\left[e_{t+1}\right] \\ &= \left(\mathbb{E}\left[u_t\right]\right)^2 \operatorname{Var}\left[h_1\right] + \left(\mathbb{E}\left[h_1\right]\right)^2 \operatorname{Var}\left[u_t\right] + \operatorname{Var}\left[u_t\right]\operatorname{Var}\left[h_1\right] + \\ &+ u_{[t-1,t-n-1]}^T \Sigma_{h_{[2,n]}} u_{[t-1,t-n-1]} + \sigma^2 \\ &= \left(\pi\left(\varphi_t, \bar{\theta}, \hat{y}_t\right)\right)^2 \Sigma_{h_{1,1}} + \hat{h}_1^2 \Pi_t^T \Sigma_t^y \Pi_t + \Pi_t^T \Sigma_t^y \Pi_t \Sigma_{h_{1,1}} + \\ &+ u_{[t-1,t-n-1]}^T \Sigma_{h_{[2,n]}} u_{[t-1,t-n-1]} + \sigma^2\end{aligned} \tag{5.24}$$

where $\Sigma_{h_{[i,j]}}$ is the square matrix retrieved form Σ from position (i,i) to position (j,j).

By iterating this process, all future output variables can be characterized by the Gaussian approximation of their distributions. However the true quantity of interest in order to retrieve a probability distribution for the return are the costs, since its value is given by the lump sum of the latter over the time horizon T.

The computation of the expectation is straightforward, while for the variance can be helpful to resort again to a first-order Taylor expansion, result-

ing in:

$$c_t \approx (y_t - r_t)^2\big|_{y_t=\hat{y}_t} + \left.\frac{\partial c_t}{\partial y_t}\right|_{y_t=\hat{y}_t} (y_t - \hat{y}_t) \tag{5.25}$$

that yields the approximation for both mean and variance:

$$\mathbb{E}[c_t] = (\hat{y}_t - r_t) \tag{5.26}$$

$$\mathrm{Var}[c_t] = 4(\hat{y}_t - r_t)^2 \Sigma_t^y \tag{5.27}$$

Being this derivation easy to be generalized for future time instants by iterating the procedure above, the final expectation for the return will be given by:

$$\mathbb{E}[J \mid \varphi_t] = \sum_{t=t_0}^{t_0+T-1} \mathbb{E}[c_t] \tag{5.28}$$

Finally, assuming all the costs at different time instants as uncorrelated from each other, an approximation of the variance of the return is given by:

$$\mathrm{Var}[J \mid \varphi_t] = \mathrm{Var}\left[\sum_{t=t_0}^{t_0+T-1} c_t\right] \approx \sum_{t=t_0}^{t_0+T-1} \mathrm{Var}[c_t] \tag{5.29}$$

completing the second-order description of its distribution.
If this density is approximated as a Gaussian, then the computed quantities are enough to have a full picture of the sought probability distribution, that can be used in order to weight models as a penalty term. However, the true density of the return is quite far from being a Gaussian, given for instance that it is always positive being a quadratic form.

5.2.1 Setting and Simulations

As it has been highlighted, in order to carry on the procedure outlined above it is necessary to first fix a value for θ, since there is the need to compute the input u_t given the output y_t through the policy, that is θ-dependent. Therefore the result is an approximation of the probability distribution for J following a fixed policy $\bar{\pi} = \pi(\varphi_t, \bar{\theta}, y_t)$.

Recall that the main goal of the overall algorithm is to find the optimal parameter θ^* that achieve the lowest possible cost in the time horizon T considered. However, in this first stage it is just required that the distribution of J is well approximated for any value of θ. If this is the case, it is possible to run the procedure for a fine grid of values for θ and thus retrieve a reasonable prior to set up the regularization framework previously described. The most important thing is that this procedure does not involve any further interactions with the unknown system, apart from those already carried out for the estimation of the state-transition function. Therefore the cardinality of the set of values for θ in which the distribution of J will be evaluated is just a computational matter.
Simulations are needed to validate the meaningfulness of the derivation outlined above. Indeed, despite its formal correctness, the computations rely on strong assumptions and approximations that may heavily affect the final outcome, hence there are no guarantees to obtain a suitable description of the real distribution. Nonetheless, it would be enough to characterize the main shape of this distribution, in order to make it useful in actual implementations of the algorithm. Clearly it is possible that the practical relevance of this approach depends highly on the considered setting.
In what follows a very simple framework will be considered, consisting of a first test to validate the procedure.
For the true system it is considered an input-output relation with discrete transfer function equal to:

$$G(z) = \frac{0.28261z^{-3} + 0.50666z^{-4}}{1 - 1.41833z^{-1} + 1.58939z^{-2} - 1.31608z^{-3} + 0.88642z^{-4}} \tag{5.30}$$

which is BIBO stable.
The function is estimated as a FIR filter

$$y_t = F(z)\, u_{t-1} + e_t \tag{5.31}$$

by means of kernel-based PEM method using the TC kernel.
500 realizations of a Gaussian white noise are used as inputs to obtain the input-output pairs for the system identification procedure. The variance of

the noise at the output channel is set to $\sigma^2 = 0.1$.
The policy under study is a simple proportional controller given by

$$\pi(\phi_t, \theta, y_t) = \theta(r_t - y_t) = u_t \tag{5.32}$$

while the time horizon is set to $T = 10$, and the initial condition is fixed at 0.
The controller should try to stabilize the system to a Heaviside step function, i.e. $r_t = \delta_{-1}(t)$.
The distribution of the overall cost J is assumed to be Gaussian, with its mean and variance computed as previously described, and it is compared with the discrete distribution obtained by several Monte Carlo trials. In every trial it is considered a different realization of the state-transition function and the system is propagated assuming that realization as the true model: the resulting overall cost is seen as a realization of the distribution of J. By extracting many samples it is possible to gain a reasonable description of the underlying distribution.
From Fig. 5.1 it is possible to see that the Gaussian approximation of the cost function does not yield good performances for any value of θ.
However the Monte Carlo sample reveal that the shape of the distribution is not Gaussian at all, as one could expect.
Given the quadratic form of the cost, another assumption on the shape of the distribution has been considered: by taking a non-central X^2 distribution the results get better (Fig. 5.2).
However for many values of the parameter θ the approximation is still very poor. Therefore in proposing the mixed approach between model-based and model-free perspective, the Monte Carlo approximation will be considered.

5.3 The main procedure

In this section the actual algorithm will be proposed in its entirety, and it will be then compared with both model-based and model-free perspectives.
With the purpose of eluding the issues raised by the computation and use

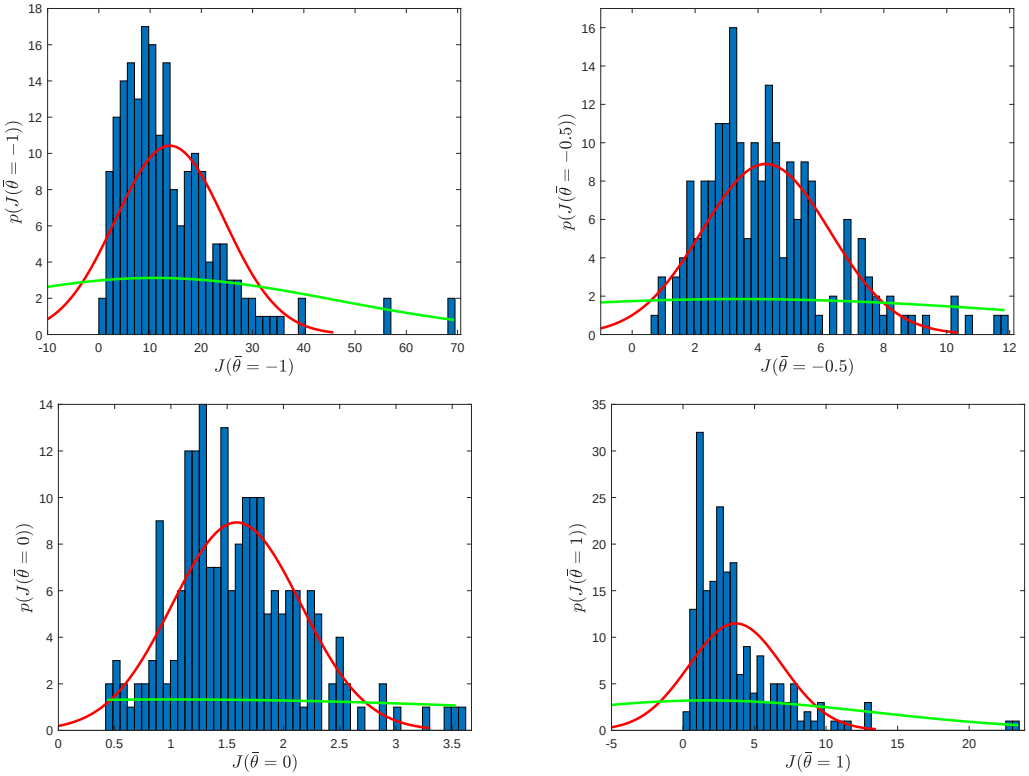

Figure 5.1: In figures above are depicted the Monte Carlo distribution in blue, its best Guassian approximation in red and the actual Gaussian approximation from the procedure in green, for $\theta = [-1, -0.5, 0, 1]$

of an approximation of the distribution of J, another approach is here presented, that exploit Monte Carlo principle.
Monte Carlo methods are a broad class of computational algorithm that rely on repeated sampling in order to obtain numerical results. The central idea is to collect enough data to have a good understanding of the distribution from which they have been generated, or at least one of its specific features. The simplest example of a Monte Carlo method is given by the sample mean, which is the most intuitive estimator for the expectation of a distribution, that consist in averaging all collected samples. The law of large numbers ensures that the sample mean converges to the true value of the expectation as the number of samples tends to infinity.
Monte Carlo approach can be used as an alternative to approximate the distribution of J coming from the model estimated through system identifi-

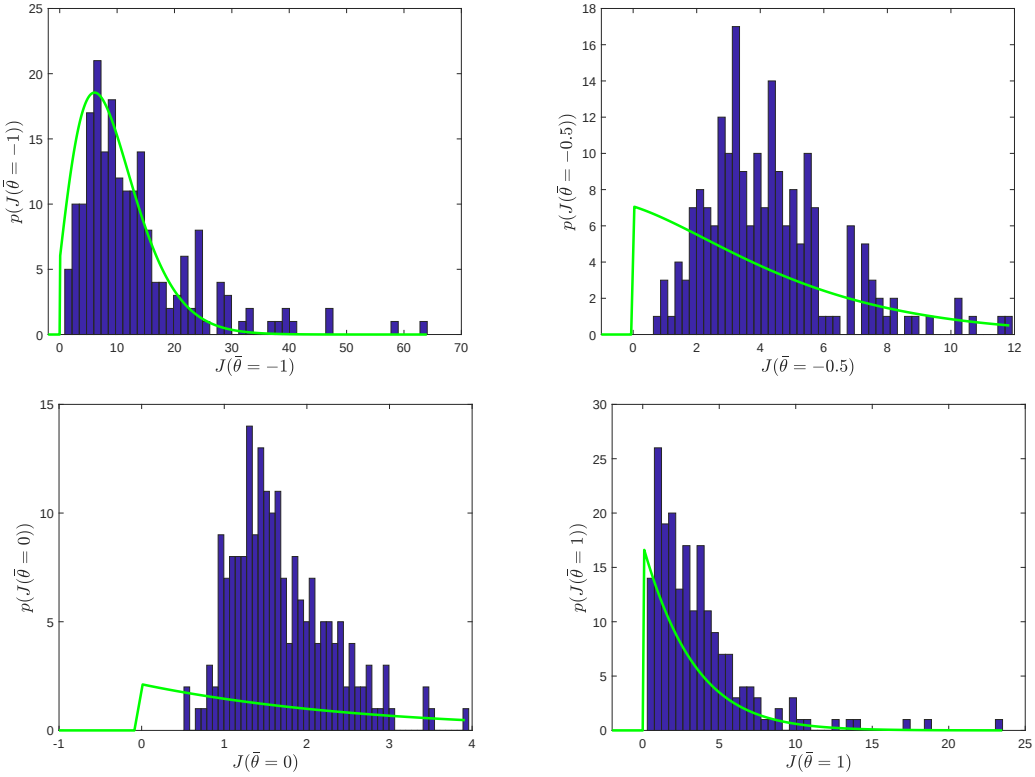

Figure 5.2: In figures above are depicted the Monte Carlo distribution in blue and the actual X^2 approximation from the procedure in green, for $\theta = [-1, -0.5, 0, 1]$

cation. That is, instead of computing the probability density of the return as a function of the policy parameter θ, that result in a poor approximation of the true distribution, the latter is sampled a large number of times and these samples are used as a regularizator.

Indeed the regularization term should reflect an inductive bias over a return function that is likely to match the state-transition mapping learned from data collected exiting the system: this can be done either through a penalty term that penalizes return function that do not represent a good match for the estimated model, or through data sampled from the return function built over the estimated model that should be fitted by the reconstructed function. This results in the same bias towards a certain class of models, as required by the problem.

The samples from the return distribution can be obtained by extracting a realization of the model following the learned density, and using that model

to compute the return value for a fixed θ in a deterministic way. This leads to a wrong bias of the framework towards the sampled model, but extracting many samples the results will reflect the true distribution and so the real behaviour of the return.
This is the main contribution of this Master Thesis, i.e. an algorithm that mixes both approaches and avoid cumbersome computation exploiting regularization.
Keeping the same framework as in the previous section, the model is given by (5.1) and the goal is to minimize the final cost over the time horizon T, given by (5.2), for a generic policy depending on the parameter vector θ.
Consider then having the possibility of querying the system only for a certain number of times, N_{tot}: these should be devoted either to run the system identification procedure following the model-based perspective, or to interact with the system through a specific policy and evaluating its performance, as required by mode-free methods. The proposed approach instead suggests to split the interactions with the real system and to obtain two different dataset: one will be used for retrieving a rough knowledge of the system and the other to test some particular values of interest for the parameter that defines the policy.
Hence it is required to assign to the system identification procedure a fixed number of samples, N_{mb}, while leaving to the policy evaluation strategy the other $N_{mf} = N_{tot} - N_{mb}$. Again it should be noticed that obtaining the punctual estimate for a specified policy involves interactions over the entire time horizon considered. When the number of data is very limited this leads to the constraint of having just a few samples to rely on when trying to retrieve the cost function over θ.
As a further matter, by keeping the initial condition fixed, the framework considered becomes too restrictive: what should be required instead is that performances are satisfactory for a large set of initial states of the system under study. In order to generalize the analysis for a random initial position, a very long time horizon is taken into account, and every sequential subset of length T is regarded as a different outcome from the true system driven by the same policy, with different initial conditions.

Therefore the cost generated for a certain policy would be:

$$\begin{bmatrix} c_{t_0} & c_{t_1} & c_{t_2} & \dots & c_{t_L} \end{bmatrix} \tag{5.33}$$

with $L >> T$ thus providing a total of $L - T + 1$ different system trajectories relying on the same policy but starting every time from a different initial condition.
This procedure allows to mediate over initial states, while keeping as small as possible the number of interaction with the system.
In the end the splitting will yield N_{mb} input-output pairs for the system identification procedure to estimate the state-transition function, and $N_{tr} = N_{mb}/L$ punctual evaluation of the cost in correspondence with different values of the parameter θ.
The result of the estimation process is a a probability distribution over input-output mappings $h \sim (\hat{h}, \Sigma_h)$, here taken as Gaussian. This distribution is then sampled N_{MC} times, with N_{MC} large enough in order to explore widely the density of model. For each realization $\check{h}$, the corresponding return $J(\bar{\theta})$ is then computed for a given value of the policy parameter $\bar{\theta} \in \Theta$, averaging over initial condition as explained above. Having removed all the randomness by relying on a certain realization for the state-transition function, all the computation can be carried out in a deterministic framework. This is iterated for all the values of interest of the parameter θ. Hence, for each value of θ on a grid that should be chosen beforehand, N_{MC} different estimates of J are found, already averaged over the initial condition, that represent the discrete distribution of the cost for that specific θ. Therefore it is possible to compute the sample mean and the sample variance of that distribution: these two values will represent the punctual estimation of the cost and its reliability, respectively.
In summary:

$$\begin{aligned} \hat{J}^{mb}\left(\hat{\theta}\right) &= \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \tilde{J}_{\check{h}}\left(\hat{\theta}\right) \\ \Sigma_J^{mb}\left(\hat{\theta}\right) &= \frac{1}{N_{MC}-1} \sum_{i=1}^{N_{MC}} \left(\hat{J}^{mb} - \tilde{J}_{\check{h}}\right)^2 \end{aligned} \tag{5.34}$$

where

$$\tilde{J}_{\check{h}} = \frac{1}{L-T+1} \sum_{i=1}^{L-T+1} \sum_{t=t_{i-1}}^{t_{i-1}+T-1} c_t^{\check{h}} \left(\hat{\theta}\right) \tag{5.35}$$

being $\check{h}$ a realization of the distribution h over models.
The second part is related to the model-free approach, and indeed consists in fixing $\theta = \bar{\theta}$ and acting on the real system with the control drawn from the policy $\pi(\varphi, \bar{\theta}, y_t)$ for the whole time horizon L. The resulting values for the return are the most reliable data for the later reconstruction of the return function, in which the only uncertainty comes from the noise at the output channel e_t. The estimates of the cost are given by averaging the obtained costs over the initial conditions, and a measure of the reliability is given again by the sample variance.

$$\begin{aligned} \hat{J}^{mf}\left(\hat{\theta}\right) &= \frac{1}{L-T+1} \sum_{i=1}^{L-T+1} \sum_{t=t_{i-1}}^{t_{i-1}+T-1} c_t^{h_0}\left(\hat{\theta}\right) \\ \Sigma_J^{mf}\left(\hat{\theta}\right) &= \frac{1}{L-T} \sum_{i=1}^{L-T+1} \left(\hat{J} - \sum_{t=t_{i-1}}^{t_{i-1}+T-1} c_t^{h_0}\left(\hat{\theta}^{mf}\right) \right)^2 \end{aligned} \tag{5.36}$$

where h_0 represent the true model.
Finally, the reconstruction of the cost function over θ should rely on both kind of data. Since it is reasonable to assume that by continuously varying the parameter, the cost will also vary continuously, a Gaussian process is considered for the reconstruction. Moreover Gaussian process models allows to weight every punctual reconstruction in a different way, according to its measure of reliability.
Depending on the problem at hand, different splitting ad different choices of the grid of values for θ can be adopted, making the proposed approach more flexible with respect to both procedures taken individually.
In the next section, an example is presented, which highlights the benefits of the new method.

5.3.1 A prototypical example

In this section the comparison between the strategies discussed so far will be analysed, with reference to a particular example.
Consider as the true model a fourth-order system, given by:

$$G(z) = \frac{0.28261z^{-3} + 0.50666z^{-4}}{1 - 1.41833z^{-1} + 1.58939z^{-2} - 1.31608z^{-3} + 0.88642z^{-4}} \tag{5.37}$$

which is BIBO-stable.
The total number of interactions available to deal with the system is fixed to $N_{tot} = 200$.
Considering a time horizon of $T = 10$ time-steps and the length of the whole trajectory of costs generated when querying the system as $L = 50$, is straightforward to see that model-free strategies are disadvantaged because they can operate on the system a maximum of 4 times. Although precise, having so few estimates can not lead to a good reconstruction of the cost function. Indeed the aim of model-free perspective is not to gain a full reconstruction of the cost, but just to improve iteratively the policy parameter.
On the other hand, model-based perspective is not affected by the choice of T and L. Once the model has been determined, every further computation can be carried out no longer by looking at the true system, but just at the estimated one. This can be a successful approach, provided the system identification procedure outputs a good model, and that is not always the case.
The policy considered in this example is a simple integrator given by

$$\pi(\phi_t, \theta, y_t) = \theta(u_{t-1} + r_t - y_t) = u_t \tag{5.38}$$

and the reference is still taken constant and equal to a Heaviside step function, i.e. $r_t = \delta_{-1}(t)$.
The variance of the noise at the output channel is equal to $\sigma^2 = 0.01$.
The true cost function depicted in Fig. 5.3 is evaluated by querying the actual system with a fine grid of values for θ.
As it can be verified through standard computations, the interval for θ that guarantees the BIBO stability of the system in closed loop is given by ap-

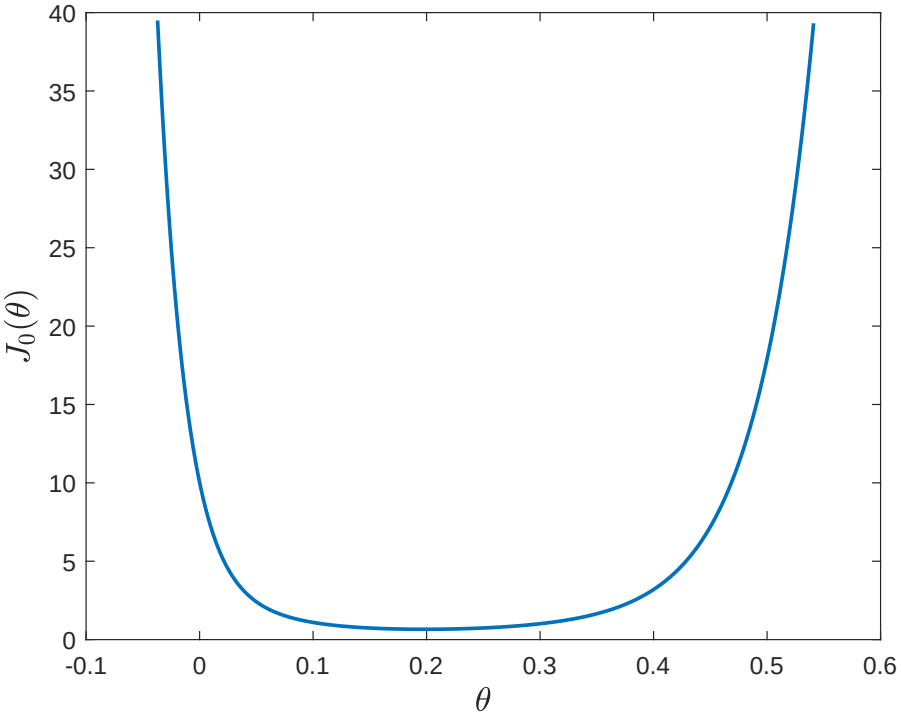

Figure 5.3: The true cost function evaluated for values of θ around the stability interval

proximately $\theta \in [0, 0.5]$. In order to make the comparison more understandable, both the values of θ for the model-based approach and the ones for the model-free perspective have been taken within that interval.
As it has been said before, only four different kind of splitting are possible:

- *Model-based*: 200 data are used to retrieve the model, and no direct interaction with the system is considered
- *Mixed approach 1*: 150 input-output pairs for the model, and just 1 query on the real system
- *Mixed approach 2*: 100 data for the model, and 2 different values of θ actually tested on the system
- *Mixed approach 3*: 50 data assigned to the model, 3 queries on the true system
- *Model-free*: No model is considered, only 4 direct interaction with the system

Models are estimated as ARX model by means of kernel-based PEM method using the TC kernel. The input is always taken as a Gaussian white noise. For the comparison, the grid of values for θ in the model-based approach is set to $[0 : 0.1 : 0.5]$ while the queries on the system are performed sampling θ from a uniform distribution in $[0, 0.5]$.
The reconstruction is performed by resorting to a Gaussian Process model, with a high prior mean in order not to allow the minimum point to fall far apart from the expected location.

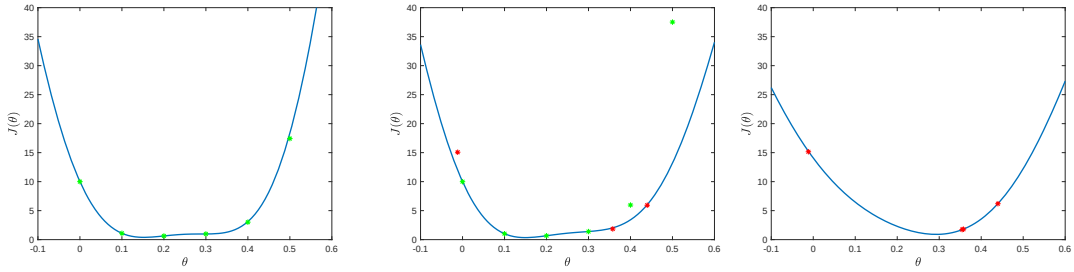

Figure 5.4: In figures above are depicted three different reconstructed cost function: the first is related to *model-based*, the second resulted from *mixed approach 3* and the last one from *model-free*

Three different reconstructed cost function taken from the same run are shown in Fig. 5.4 The result shown in Fig. 5.5 confirms the benefits of getting a rough idea of the model of the cost function, and then directly interact with the system in order to have reliable punctual measure to build a better estimate of the sought cost.
Indeed the model-free perspective here is voluntarily penalised since there is no real strategy behind the choice of the values of θ with which to query the system. However this framework allows for a fair comparison between different approaches. It is indeed clear how relying on just a few cost values makes the estimate unstable.
In any strategy that instead exploits an estimate of the model, it is possible to see that reconstructions from different runs of the whole procedure do not differ much from each other. This stable behaviour is guaranteed by the presence of the model. Moreover it is also clear that there is no need for an accurate estimate of the state-transition function: it is much more effective to leave most of the interactions for directly query the system, while having

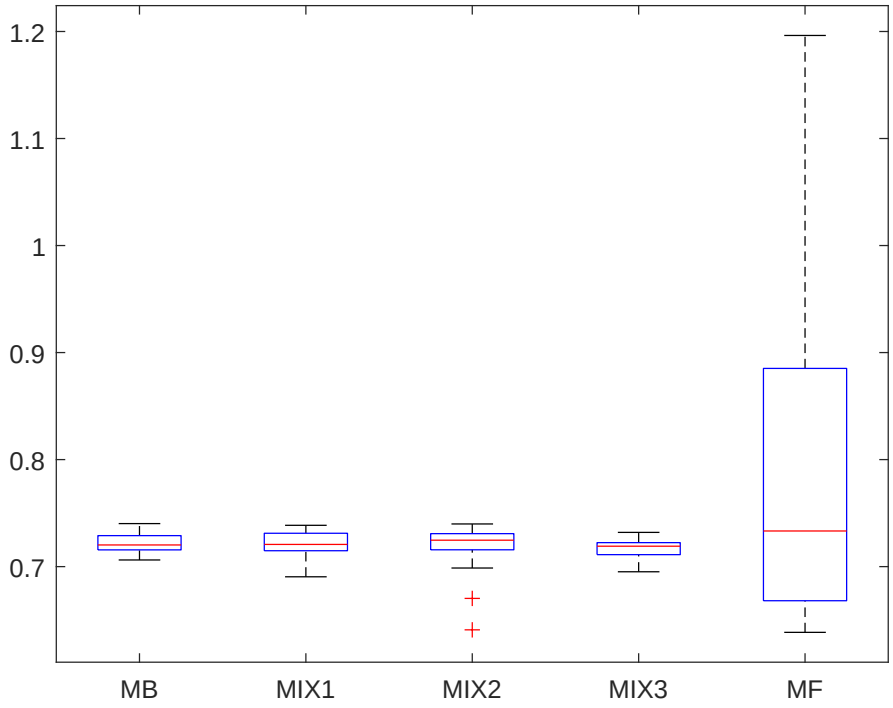

Figure 5.5: Boxplot of the performances of the different strategies. Values are obtained running a query on the true system with the value of θ that resulted the minimum point for the reconstruction with each method. The boxplots are the result of more iterations for the same procedure. It is clear how the model-free perspective has a larger variability, and how mixing the two paradigm results in a improvement from both the two perspectives

some prior knowledge over the shape of the cost function.
Indeed one of the best strategy to exploit this mixed approach would be to sample the values for θ that should be used to address the system directly from a custom distribution built upon the model estimated from the model-based step. This would allow the function to be refined more likely where the cost is low.

6
Conclusions

The different perspectives of Model-Based Reinforcement Learning and Model-Free Reinforcement Learning have been deeply analysed and reviewed. Although the two framework seem very different as Reinforcement Learning paradigm, restating the problem in the Optimal Control framework provided a tool for mixing the two opposed scenarios.
Model-free methods completely neglect the model of the environment, while searching for a direct mapping form policy to reward. Model-based strategies instead put all of their efforts in trying to retrieve the best possible model for the environment, in order to move the learning paradigm into System Identification.
However this two approaches can be linked through regularization. Indeed from the model of the environment it is possible to generate data for the estimation of the cost function by means of Monte Carlo procedures. This kind of data are part of the same framework that involves punctual evaluations from model-free perspective.
Restated in this perspective, the two different approaches seem now complementary, since one provide a stable prior function of the cost that should be estimated, and the other gives reliable estimate which however do not yield an extensive description of the shape of the function. This leads model-free perspective to attain both good and bad scores, while model-based methods

never go below a certain level of performance, but cannot improve much their behaviour either.
In the example proposed, the mixed approach proved to be a promising way towards an algorithm that can benefit as much as possible from the peculiar features of both methods.
However, many questions remain still unanswered. First of all it would be necessary a further study on how it is possible to best exploit the knowledge of a prior shape for the cost function in order to refine the estimate in regions where the cost is low. This can be obtained by extracting θs with which to query the system from a distribution shaped accordingly to the reward function (or inversely to the cost). However this strategy should be validated through simulation, and it lacks the fundamental component of exploration in RL. Moreover, the prototypical example suggested the need for a lot of queries and the help of a low-accuracy model: it would be interesting to understand whether it was possible to have an indication of the best splitting based only on a part of the available interactions, and then to distribute the remaining ones more effectively. Lastly, it is required to evaluate the effectiveness of the presented approach in a multi-dimensional framework.

Bibliography

[1] F. Berkenkamp, M. Turchetta, A. P. Schoellig, A. Krause. Safe Model-based Reinforcement Learning with Stability Guarantees. *31st Conference on Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA, 2017.

[2] M. P. Deisenroth, C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the* 28^{th} *International Conference on Machine Learning*. Bellevue, WA, USA, 2011.

[3] E. Ipek, O. Mutlu, J. F. Martinez, R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In ISCA'08:*Proceedings of the 35th Annual International Symposium on Computer Architecture*, pp. 39–50. IEEE Computer Society Washington, DC, USA, 2008.

[4] G. Picci. *Filtraggio Statistico (Wiener, Levinson, Kalman) e applicazioni*. Edizioni Libreria Progetto Padova, 2007.

[5] B. Recht. A Tour of Reinforcement Learning: The View from Continuous Control. arXiv:1806.09460v2, 2018.

[6] D. Romeres, M. Zorzi, R. Camoriano, S. Traversaro, A. Chiuso. Derivative-free online learning of inverse dynamics models. arXiv:1809.05074, 2018.

[7] U. Rosolia, F. Borrelli. Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework. In *IEEE Transaction on Automatic Control*, Vol. 63, no. 7, pp 1883-1896, 2018.

[8] S. Shalev-Shwartz, S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

[9] R. S. Sutton, A. G. Barto. *Reinforcement Learning - An Introduction*. The MIT Press, Cambridge, Massachusetts, MA, II edition, 2018.

[10] M. Zorzi. Lecture Notes in System Identification, 2011.