Université de Liège

Faculté des Sciences Appliquées

Closed-Loop Learning of Visual Control Policies

(Preliminary version of September 26, 2006)

Thèse de doctorat présentée par

Sébastien Jodogne

en vue de l'obtention du grade de
Docteur en Sciences (orientation Informatique)

Année académique
2006 - 2007

ABSTRACT

In this dissertation, I introduce a general, flexible framework for learning direct mappings from images to actions in an agent that interacts with its surrounding environment. This work is motivated by the paradigm of purposive vision. The original contributions consist in the design of reinforcement learning algorithms that are applicable to visual spaces. Inspired by the paradigm of local-appearance vision, these algorithms exploit specialized visual features that can be detected in the visual signal.

Two different ways to use the visual features are described. Firstly, I introduce adaptive-resolution methods for discretizing the visual space into a manageable number of perceptual classes. To this end, a percept classifier that tests the presence or absence of few highly informative visual features is incrementally refined. New discriminant visual features are selected in a sequence of attempts to remove perceptual aliasing. Any standard reinforcement learning algorithm can then be used to extract an optimal visual control policy. The resulting algorithm is called *Reinforcement Learning of Visual Classes*. Secondly, I propose to exploit the raw content of the visual features, without ever considering an equivalence relation on the visual feature space. Technically, feature regression models that associate visual features with a real-valued utility are introduced within the Approximate Policy Iteration architecture. This is done by means of a general, abstract version of Approximate Policy Iteration. This results in the *Visual Approximate Policy Iteration* algorithm.

Another major contribution of this dissertation is the design of adaptive-resolution techniques that can be applied to complex, high-dimensional and/or continuous action spaces, simultaneously to visual spaces. The *Reinforcement Learning of Joint Classes* algorithm produces a non-uniform discretization of the joint space of percepts and actions. This is a brand new, general approach to adaptive-resolution methods in reinforcement learning that can deal with arbitrary, hybrid state-action spaces.

Throughout this dissertation, emphasis is also put on the design of general algorithms that can be used in non-visual (e.g. continuous) perceptual spaces. The applicability of the proposed algorithms is demonstrated by solving several visual navigation tasks.

ACKNOWLEDGMENTS

Tout d'abord, je souhaite exprimer toute ma gratitude envers Monsieur J. Piater, promoteur de cette thèse, qui m'a accueilli et m'a proposé un sujet riche et passionnant. Je lui suis notamment extrêmement reconnaissant pour toute la confiance qu'il m'a témoignée en me donnant une grande liberté dans mes recherches. Qu'il soit aussi remercié pour sa gentillesse, pour ses idées précieuses, pour son respect et bien sûr pour l'encadrement scientifique apporté au cours de ces trois années.

Je remercie Messieurs V. Charvillat, R. Munos, L. Paletta, J. Verly et L. Wehenkel, membres du jury, pour avoir accepté de consacrer du temps à la lecture et à l'évaluation de ce document.

Je suis également extrêmement reconnaissant envers le FNRS et Monsieur P.-A. de Marneffe pour m'avoir fourni le support financier nécessaire à la réalisation de mon doctorat.

Un de mes plus proches amis mérite une place toute particulière dans ces remerciements. Il s'agit de Cyril Briquet. Qu'il soit bien sûr remercié pour tout le temps passé à la relecture de cette thèse, pour tous les précieux conseils prodigués et pour notre publication commune. Mais en outre, je lui serai toujours reconnaissant pour le *coaching* psychologique diablement efficace dont j'ai pu bénéficier. Merci aussi à Elisabeth Spilman pour sa relecture de l'introduction, ce qui lui a donné plus de consonances anglophones.

Je tiens également à remercier l'ensemble du personnel académique, scientifique, technique et ouvrier de Montefiore qui m'a permis de construire mon doctorat dans une ambiance conviviale et agréable. Je ne connais pas la moitié d'entre vous à moitié autant que je le voudrais. Mes plus vifs remerciements vont notamment à Olivier Barnich, Axel Bodart, Danielle Borsu, Christophe Burnotte, Renaud Dardenne, Fabien Defays, Gérard Dethier, Damien Ernst, Jean-Marc François, Christophe Germay, Pierre et Ibtissam Geurts, Xavier Hainaut, Jean-Bernard Hayet, Frédéric Herbreteau, Benoît Jaspart, Axel Legay, Jean Lepropre, Raphaël Marée, Sylvain Martin, Fabien Scalzo, Hugues Smeets, Guy-Bart Stan et Cédric Thiernesse. Chacun de vous a contribué, sans doute sans s'en apercevoir, à bien des lignes de cette thèse, que ce soit par votre aide, votre attention ou tout simplement votre sourire. Merci aussi à toute l'équipe du service de vision par ordinateur. Je m'excuse par la présente auprès de tous les étudiants de Montefiore pour les cycles CPU goulûment volés par mes calculs distribués, ainsi qu'auprès des *switches* du SEGI pour les souffrances sans nom qu'ils ont dû endurer par ma faute.

Durant ces cinq années, j'ai pu compter sur l'affection et l'attention de nombreux amis. Je citerai notamment Raffaelle et Amandine Brancaleoni, Manuel Gérard, Pierre et Lisiane Holzemer, Fabian et Laurence Lapierre, Michel et Florence Prégardien et Lara Vigneron. D'un point de vue personnel, le soutien et l'écoute de l'équipe d'animation de Dalhem ont été capitaux dans certains moments plus difficiles. Merci tout particulièrement à Céline Lambrecht, Dominique Olivier, Maurice Simons et Benoît Vanhulst. Je pense aussi à l'ensemble de notre équipe de cheminement *Amour et Engagement*, ainsi qu'à l'équipe d'animation de Visé pour la préparation au mariage. Merci à vous... et, bien sûr, merci à tous ceux que j'ai oublié de remercier ! Qu'ils m'en excusent ;-)

Ce travail n'aurait pas non plus été possible sans la présence à mes côtés de ma famille et de ma belle-famille. Je pense aujourd'hui tout particulièrement à mon grand-père, qui m'a ouvert à la curiosité et au questionnement scientifiques et qui m'a initié à l'informatique.

Enfin, tout mon amour et toute ma tendresse reviennent à mon épouse Delphine pour ses encouragements, sa patience et sa présence tout au long de mes (longues) études. Je t'aime !...

Lanaye, le 26 septembre 2006.

A la mémoire de Pépé

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

ABBREVIATIONS AND NOTATION

Acronyms and abbreviations:

API	Approximate Policy Iteration
BDD	Binary Decision Diagram [Bry92]
DoG	Difference of Gaussians (interest point detector) [Low99]
LSPI	Least-Squares Policy Iteration [LP03]
MDP	Markov Decision Process
FMDP	Finite Markov Decision Process (state and action spaces are finite)
RL	Reinforcement Learning [BT96, KLM96, SB98]
RLJC	Reinforcement Learning of Joint Classes [JP06b]
RLVC	Reinforcement Learning of Visual Classes [JP05c]
SIFT	Scale-Invariant Feature Transform (local descriptor) [Low04]
TD	Temporal Difference [Sut88]
V-API	Visual Approximate Policy Iteration [JBP06]
RETQ	Raw Extra-Tree state-action value function approximator

General notation:

$\cdot \mapsto \Pi(\cdot)$	probabilistic relation
$\mathcal{B} = \{\mathbf{true}, \mathbf{false}\}$	set of Boolean numbers
$G(\mu, \sigma)$	Gaussian law of mean μ and standard deviation σ
$\mathcal{P}(\cdot)$	power set
$\mathbb{N}$	set of positive integers
$\mathbb{R}$	set of real numbers
$\mathbb{R}^+$	set of positive real numbers
$\mathbb{R}_0^+$	set of positive, non-zero real numbers
$\boldsymbol{x}$	vector

Notation for reinforcement learning:

$\pi : S \mapsto \Pi(A)$	percept-to-action mapping (i.e. probabilistic, Markovian, stationary control policy)
π^*	optimal percept-to-action mapping
A	action space (also known as control space)
H	Bellman backup operator (cf. Equation 2.26)
H^π	Bellman backup operator for control policy π
$Q^\pi(s, a) : S \times A \mapsto \mathbb{R}$	state-action value function of policy π
$Q^*(s, a) : S \times A \mapsto \mathbb{R}$	optimal state-action value function
$\mathcal{R}(s, a) : S \times A \mapsto \mathbb{R}$	reinforcement signal
S	state space, possibly a set of images
$\langle S, A, \mathcal{T}, \mathcal{R} \rangle$	Markov Decision Process
$\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$	interaction
$\mathcal{T}(s, a, s') : S \times A \mapsto \Pi(S)$	probabilistic transition relation
T	Bellman backup operator (cf. Equation 2.11)
T^π	Bellman backup operator for control policy π
$V^\pi(s) : S \mapsto \mathbb{R}$	value function of control policy π
$V^*(s) : S \mapsto \mathbb{R}$	optimal value function

Notation for computer vision:

$d(\boldsymbol{x}, \boldsymbol{y})$	metric over the visual feature space
$\mathcal{D}_L : S \mapsto \mathcal{P}(\mathbb{R}^2 \times W)$	interest point detector (cf. Definition 3.3)
$\mathcal{G}_L : S \times (\mathbb{R}^2 \times W) \mapsto V$	local descriptor generator (cf. Definition 3.5)
$\mathcal{G}_V : S \mapsto \mathcal{P}(V)$	visual feature generator (cf. Definition 3.1)
I_x (resp. I_y)	gradient over the x-axis (resp. y-axis) of an image I smoothed by a Gaussian
I_{xx}, I_{xy}, I_{yy}	second-order derivatives of a smoothed image
V	visual feature space (usually corresponds to $\mathbb{R}^n$)
W	support window space (cf. Definition 3.3)

Notation for RLVC, V-API and RLJC:

Δ_t	Bellman residual, also known as the temporal difference at time t (cf. Equation 2.44)
$\mathcal{A} : \mathcal{P}(F_A) \times \mathcal{P}(F_A) \mapsto A$	action seeker (cf. Definition 7.8)
$B\Big\rvert_{\boldsymbol{b}}$	application of a BDD B to a vector of Booleans $\boldsymbol{b}$
$\mathcal{B}(n)$	family of BDDs with n Boolean inputs
$c_i^{(k)}$	perceptual or joint class induced by $\mathcal{C}_k$ or by $\mathcal{J}_k$
C_k	space of joint classes induced by $\mathcal{J}_k$
$\mathcal{C}_k : S \mapsto S_k$	percept classifier (cf. Definition 4.6)
$\mathcal{D} : (S \times A) \times F \mapsto \mathcal{B}$	joint feature detector
$\mathcal{D}_A : A \times F_A \mapsto \mathcal{B}$	action feature detector (cf. Definition 7.1)
$\mathcal{D}_V : S \times V \mapsto \mathcal{B}$	visual feature detector (cf. Definition 4.3)
$\mathcal{D}_S : S \times F_S \mapsto \mathcal{B}$	perceptual feature detector (cf. Definition 4.4)
$\mathcal{F}$	class of state-action value function approximators
$F = F_S \cup F_A$	joint feature space
F_A	action feature space
F_S	perceptual feature space
$\mathcal{G} : \mathcal{P}(S \times A) \mapsto \mathcal{P}(F)$	joint feature generator
$\mathcal{G}_A : \mathcal{P}(A) \mapsto \mathcal{P}(F_A)$	action feature generator (cf. Definition 7.2)
$\mathcal{G}_R : S \mapsto \mathcal{P}(\mathbb{R}^n)$	raw feature generator (cf. Definition 6.1)
$\mathcal{G}_S : \mathcal{P}(S) \mapsto \mathcal{P}(F_S)$	perceptual feature generator (cf. Definition 4.5)
$\mathcal{J}_k : S \times A \mapsto J_k$	joint classifier (cf. Definition 7.6)
$\mathcal{L} : \mathcal{P}(\mathcal{S} \times \mathcal{A} \times \mathbb{R}) \mapsto \mathcal{F}$	learning oracle (cf. Definition 6.3)
S_k	space of perceptual classes induced by $\mathcal{C}_k$

CHAPTER
ONE

Introduction

Designing robotic controllers rapidly proves to be a challenging problem. Indeed, such controllers (1) face a huge number of possible inputs that can be noisy, (2) must select actions among a continuous set, and (3) should be able to automatically adapt themselves to evolving or stochastic environmental conditions. Although a real-world robotic task can often be solved by directly connecting the perceptual space to the action space through a given computational mechanism, such mappings are usually hard to derive by hand, especially when the perceptual space contains images. Thus, automatic methods for generating image-to-action mappings are highly desirable because many robots are nowadays equipped with CCD sensors.

This class of problems is commonly referred to as vision-for-action*. Living beings face vision-for-action problems everyday and* learn *how to solve them effortlessly and robustly. Despite roughly three decades of research, vision-for-action is still a major challenge in both computer vision and artificial intelligence. A potentially fruitful research path for learning visual control policies*[1] *would therefore be to mimic natural learning strategies. This dissertation introduces a general framework that is suitable for building image-to-action mappings using fully automatic and flexible learning protocols that resort to* reinforcement learning*.*

1.1 Vision-for-Action

In this dissertation, I am interested in reactive systems that learn to couple visual perceptions and actions inside a dynamic world so as to act reasonably. This coupling is known as a *visual (control) policy*. This wide category of problems will be called *vision-for-action tasks* (or simply *visual tasks*). Despite about fifty years years of active research in artificial intelligence, robotic agents are still largely unable to solve many real-world visuomotor tasks that are easily performed by humans and even by animals. Such vision-for-action tasks notably include grasping, vision-guided navigation and manipulation of objects so as to achieve a goal.

[1] The terms “visual control policy” and “image-to-action mapping” are used indistinctly.

1.1.1 Reconstructionist Vision

The traditional *reconstructionist approach* to computer vision is often taken into consideration when solving visual tasks. This paradigm consists in generating a complete, detailed, symbolic 3D model of the surrounding environment [Mar82a]. According to this paradigm, originally proposed by Marr, a computer vision system is a hierarchy of bottom-up processes. Each module in this hierarchy transforms information from an abstraction level into information belonging to a higher abstraction level. Each abstraction level comes with its own representation system for encoding the information. The bottom level (the *primal sketch*) takes raw pixels as input, and generally consists either in the *segmentation* of the image into regions of interest or in the extraction of *visual features* (cf. Chapter 3). The top level ultimately uses the symbolic representation of the environment so as to choose how to react.

Reconstructionist vision had and still has a great influence on research in computer vision. However, it has also been the subject of important criticism:

1. In practice, it is impossible to reconstruct a fully elaborated representation of a visual scene, notably because the projection of a 3D scene into a 2D image causes information loss.

2. Reconstructionist vision systems are rigid, to wit, they are often limited by design to one particular visual task. Moreover, most systems are designed to operate on a fixed set of scenes or objects.

3. The performed task does not weigh in the way the high-level symbolic representation is built. Thus, even when the model of the environment is computed, a decision making process that may be complex is still needed to choose the optimal reaction.

4. The tasks to be solved are explicitly coded inside the system. Such a task specification is generally difficult to express formally and unambiguously.

5. Many reconstructionist systems are hard-wired and/or do not take lessons from their prior history, making them unable to improve over time or to adapt to novel situations that were not anticipated by their designers. Many systems also operate under highly controlled conditions and in a fixed context (e.g. exclusively indoor or outdoor, with a known background, and/or without clutter).

These criticisms have motivated the development of alternatives to the reconstructionist paradigm [CRS94]. These alternatives are based on a study of the neuropsychological development of the human visual system. The development of sensory-motor coordination of newborns, especially during the first year of life, is indeed a fruitful starting point for biologically-inspired approaches to vision-for-action. Several facts about this development are outlined in the next section.

1.1.2 Human Visual Learning

An infant is not born perceiving the world as an adult would: The perceptual abilities develop over time. During the first months of life, many cognitive components are either non-functional or not yet fully developed. For example, the control of the extremities is still limited [KBTD95] and the neural growth is not completed [O'L92]. Now, strong neuropsychological evidence suggests that newborns learn to extract useful information from visual data in an interactive fashion, without any external supervisor [GS83]. By evaluating the consequences of their actions on the environment, infants learn to pay attention to visual cues that are behaviorally important for solving their tasks. When they grasp objects, smile or try and reproduce heard sounds, they become aware of the dynamics of the outside world and try to analyze the effects of their reactions in order to improve their behavior.

To this end, the behavior of newborns is optimized for gathering new experiences: It favors learning through the collection of data rather than the immediate efficiency of the reactions. Such exploratory behaviors are notably supported by reflexes [PW03]. Even noise can contribute to long-term performance by helping newborns to focus on robust, highly discriminative visual cues. In this manner, as they interact with the outside world, newborns gain more and more expertise on commonplace tasks. Moreover, human visual skills do not remain rigid after childhood: The visual system continues to evolve throughout life so as to solve unseen visual tasks, and continues to acquire expertise even on well-known tasks [TC03]. Thus, the human visual system is eminently general and adaptive.

Of course, getting representative data for acting efficiently can take a very long time depending on how vast the state space is. This can be particularly problematic in the case of visual tasks. Therefore, infants cannot settle for exploring the environment, but must at some point use the knowledge they have compiled to indeed solve the task. This issue is generally called the *exploration-exploitation trade-off* [SB98].

Obviously, the process of learning to act in presence of a visual stimulus is task-driven, since different visual tasks do not necessarily need to make the same distinctions [SR97]. As an example, consider the task of grasping objects. It has been shown that, between about five to nine months of age, infants learn to pre-shape their hands using their vision before they reach the object to grasp [MCA$^+$01]. Once the contact is made, the hand often occludes the observed object so that tactile feedback is used to locally optimize the grasp. In this context, it is clear that the haptic and the visual sensory modalities ideally complement one another. For this grasping procedure to succeed, infants have to learn to distinguish between objects that require different hand shapes. Thus, infants learn to recognize objects according to the needs of the grasping task.

Churchland et al. provide further psychological, anatomical and neuropsychological evidence for these properties of human visual learning [CRS94]. The lessons above lead to several important conclusions about the criticisms of reconstructionist vision that were drawn at the beginning of this section:

1. It is neither desirable, nor required, to generate inside the brain a fully elabo-

rated high-level representation of the observed scene. Gathering only a subset of this model might be sufficient for taking optimal decisions in a given visuo-motor task. Human beings build a *partial representation of the scene* in which only information that is relevant to the performed task is represented.

2. The human visual system is *generic* and *versatile*. Its capabilities are not limited by design, but they are generated and augmented as needed: Throughout its entire life, the visual system evolves to acquire additional visual skills.

3. The human visual system is *task-dependent*, which means that it shortcuts the visual interpretation process by delivering exactly the information that is needed for solving a given task. In this way, the task directs what information is embedded in the high-level symbolic representations in order to ease the decision making process.

4. The goal of a living being is only *implicitly defined* (e.g. feeding, fleeing, fighting, seeking novelty, experiencing discovery or the survival of the species). This definition partly resorts to pain or pleasure signals that have been hard-wired during the course of evolution.

5. The human visual system is *robust* and *adaptive*. These characteristics are closely related to the versatility of human vision. Adaptivity means that acquired visual skills are continuously tuned and improved through learning from past experience. This contributes to the improvement of performance, even on well-practiced tasks. This phenomenon is also responsible for the gain in expertise of human vision [TC03]. Expertise rules the recognition of objects at different levels of specificity based on previous experience (e.g. a biologist is able to distinguish between a monkey and a macaque).

Furthermore, it has been argued that the following important properties are inherently present in the human visual learning process:

- The learning is *interactive*, as the consequences of the reactions drive what is learned.

- The learning is *exploratory* and faces the exploration-exploitation trade-off.

1.1.3 Purposive Vision

Following from the above discussion, a breakthrough in modern artificial intelligence would be to design an artificial system that would acquire object or scene recognition skills based only on its experience with the surrounding environment. To state it in more general terms, an important research direction would be to design a robotic agent that could *autonomously acquire visual skills from its interactions* with an uncommitted environment in order to achieve some set of goals. Learning new visual skills in a dynamic, task-driven fashion so as to complete an *a priori* unknown visual task is known as the *purposive vision* paradigm [Alo90].

Purposive vision is essentially orthogonal to reconstructionist vision. It emphasizes the fact that vision is task-oriented and that the performing agent should focus its attention only on the parts of the environment that are relevant to its task. For example, if an agent has to move across a maze, it only needs to spot several interesting landmarks; It does not need to be able to recognize *all* the objects in the environment [Deu04]. Therefore, it is not required to obtain a complete model of the environment. Purposive vision stresses the dependency between action and perception: Selecting actions becomes the inherent goal of the visual sensing process. This paradigm often leads to the breakdown of the visual task into several sub-problems that are managed by a supervision module that ultimately selects the suitable reactions [Tso94].

The terminology "purposive vision" is somewhat confusing. Purposive vision is indeed very close to *active vision* [AWB88, BY92, FA95], and these concepts are sometimes used interchangeably. Just like purposive vision, active vision criticizes the passive point of view of reconstructionist vision, and it argues that visual perception is an exploratory activity. However, active vision is essentially interested in experimental setups where the position of the visual sensors can be governed by the effectors. This approach is evidently inspired by human vision, for which muscles can orientate the head, the eyes and the pupils. By learning to control such effectors, the agent can acquire better information and resolve ambiguities in the visual data, for example by acquiring images of a scene from different viewpoints. Some ill-posed problems in computer vision become well-posed by employing active vision. An instance of an active vision paradigm is Westelius' robotic arm [Wes95], which uses a focus-of-attention mechanism based on stereo vision. Active vision has also been used in visual navigation for a robot with a stereo active head [Dav98]. However, the paradigm of active vision essentially confines interactivity to the positioning of visual sensors. Thus, purposive vision is more general. Other related paradigms include *active perception* [Baj88] and *animate vision* [BB92].

1.2 Objectives

This dissertation attempts to design a general computational framework for purposive vision. More precisely, given the discussion above, my aim is to design machine learning algorithms that can cope with vision-for-action tasks, and that have the following key properties:

- *Closed-loop (interactive).* The agent should autonomously learn its task by interacting with the environment, without any help from an external teacher or from a supervisory training signal. It should be able to predict the effects of its actions, and to improve them with growing experience.

- *Task-driven.* The task to be solved should drive which visual skills are learned.

- *Minimalist.* The visual control policies that are learned should not use a complete model of the environment as in reconstructionist vision, but should rather learn to select discriminative visual cues for the task.

- *Integration of time.* An action can have a long-term effect on the environment, which must be taken into account when taking a decision. In other words, *dynamic* environments are considered.
- *Versatile.* The algorithms should apply to a broad class of vision-for-action problems, should not be tuned for one particular task, and should make few prior assumptions about the tasks and the environment. The same algorithms should be applicable in a variety of environments.
- *Implicit formulation of the goal.* The objective of the agent should not be hard wired (e.g. in a programming language), but should instead be *learned* from implicit environmental cues.

This research topic lies at the border between machine learning, computer vision and artificial intelligence. As discussed in the previous section, it can also be motivated from observations of visual neuroscience. The objectives above follow from Piater's doctoral dissertation [Pia01]. Piater indeed writes:

> "Autonomous robots that perform nontrivial sensorimotor tasks in the real world must be able to learn in both sensory and motor domains. A well-established research community is addressing issues in motor learning, which has resulted in learning algorithms that allow an artificial agent to improve its actions based on sensory feedback. Little work is being conducted in sensory learning, here understood as the problem of improving an agent's perceptual skills with growing experience. The sophistication of perceptual capabilities must ultimately be measured in terms of their value to the agent in executing its task." [Pia01]

However, Piater essentially focuses on environments in which an action does not have an impact on the long term. Every time the agent has performed an action, the system is reset, which leads to independent episodes that consists of single decisions. This category of problems is referred to as *evaluative feedback* by Sutton and Barto [SB98]. Moreover, Piater considers binary tasks, to wit, tasks in which a decision is either correct or wrong. In the framework of this dissertation, there exists a whole continuum in the appropriateness of actions, precisely because the dynamic aspect of the environment is taken into account. Just like in a chess game, a decision with immediate negative consequences (e.g. a sacrifice) may nonetheless later lead to the exploration of highly advantageous parts of the reward space (e.g. winning the game). This general problem that is inherent to dynamic environments is known as the *delayed reward problem* [SB98].

1.3 Closed-Loop Learning of Visual Policies

One plausible framework to deal with vision-for-action tasks according to purposive vision is *Reinforcement Learning* (RL) [BT96, KLM96, SB98]. Reinforcement learning is a biologically-inspired computational framework that can generate nearly optimal control policies in an automatic way, by interacting with the environment[2]. RL is founded on the analysis of a so-called *reinforcement signal*. Whenever the agent takes a decision, it receives as feedback a real number that evaluates the relevance of this decision. From a biological perspective, when this signal becomes positive, the agent experiences pleasure, and we can talk about a *reward*. Conversely, a negative reinforcement implies a sensation of pain, which corresponds to a *punishment*. Now, RL algorithms are able to map every possible perception to an action that maximizes the reinforcement signal *over time*. In this framework, the agent is never told what the optimal action is when facing a given percept, nor whether one of its decisions was optimal. Rather, the agent has to discover by itself what the most promising actions are by constituting a representative database of interactions, and by understanding the influence of its decisions on future reinforcements.

1.3.1 Motivation

Reinforcement learning is an especially attractive, well-suited paradigm for closed-loop learning of visual control policies. It is fully automatic, and it fulfills all the six objectives that were stated in Section 1.2 (interactivity, task-driven, minimalism, integration of time, versatility and implicit goal).

Indeed, RL is by definition *closed-loop* as an optimal visual control policy can be extracted only from a sequence of interactions. It is also *task-driven* because the learned policy maximizes the reinforcement signal over time, and this signal is of course task-dependent. The generated policies are *minimalist* as they directly connect a percept to a suitable reaction. The *dynamic aspect* of the environments is captured since RL does not maximize the *immediate* rewards, but rather balances immediate rewards with long-term rewards. This is the delayed reward problem that was mentioned at the end of Section 1.2. Reinforcement learning is also a highly *versatile* paradigm, as it imposes weak constraints on the environment. It can indeed cope with any problem that can be formulated in the framework of *Markov Decision Problems*. This generic category of problems will be investigated in detail in Chapter 2. Intuitively, it corresponds to dynamic environments in which the probability of reaching a state after taking an action is independent of the entire history of the system. Finally, the goal of the agent is only *implicitly encoded* as a reinforcement signal that the agent struggles to maximize.

Theoretically, it seems natural to directly incorporate image information as the perceptual information of a reinforcement learning algorithm. But this is a difficult task. Directly generating states from image results in huge state spaces, whose size exponentially grows with the size of the perceived images. As a consequence, even standard RL algorithms based on value function approximation (cf. Chapter 6) cannot deal with them without making prior assumptions on the structure of the images. Images also often contain noise and irrelevant data, which further com-

[2] As argued by Sutton [Sut04], the links between RL and Pavlovian conditioning are widely acknowledged [SB90].

plicates state construction. Furthermore, in such huge and noisy state spaces, the reinforcement signal tends to dilute. Because the perceptual space can only be very sparsely sampled, it indeed quickly becomes difficult to guess what is the property of an image that leads to high or low reinforcement. This is the well-known *Bellman curse of dimensionality*, which makes standard RL algorithms inapplicable to direct closed-loop learning of visual control policies.

The key technical contribution of this dissertation consists in the introduction of reinforcement learning algorithms that can be used when the perceptual space contains images. The algorithms that are developed do not rely on a task-specific pre-treatment. As a consequence, they can be used in any vision-for-action task that can be formalized as a Markov Decision Problem.

It should be noted that in this research, I will take an approach that may seem extremely minimalist. Indeed, no model of the environment will be built at all, and the decisions will only be taken by analyzing the visual stimulus. Thus, the proposed algorithms will learn *image-to-action mappings* that directly connect an image to the appropriate reaction. This is a direct consequence of the reinforcement learning framework.

1.3.2 Related Work

There exists a variety of work in RL about solving specific robotic problems involving a perceptual space that contains images. For instance, Schaal uses visual feedback for solving a pole-balancing task [Sch97]. RL has been used to control a vision-guided underwater robotic vehicle [WGZ99]. More recently, Kwok and Fox have demonstrated the applicability of RL for learning sensing strategies using Aibo robots [KF04]. Paletta et al. learn sequential attention models through reinforcement learning [PFS05]. I also mention the use of reinforcement learning in other vision-guided tasks such as ball shooting [ANTH94], ball acquisition [TTA99], visual servoing [GFZ00], robot docking [WWZ04, MMD05] and obstacle avoidance [MSN05]. Interestingly enough, RL is also used as a way to tune the high-level parameters of image-processing applications. For example, Peng and Bhanu introduce RL algorithms for image segmentation [PB98], whereas Yin proposes algorithms for multilevel image thresholding, and uses entropy as a reinforcement signal [Yin02].

All of these applications preprocess the images to extract some high-level information about the observed scene that is directly relevant to the task to be solved and that feeds the RL algorithm. This requires making prior assumptions about the images that will be perceived by the sensors of the agent. The preprocessing step is task-specific and is performed by hand. This contrasts with my objectives, which consist in introducing algorithms able to *learn* how to *directly* connect the visual space to the action space, without using manually written code and without relying on a priori knowledge about the task to be solved.

A noticeable exception is the work by Iida et al. who apply RL to seek and reach targets [ISS02], and to push boxes [SI03] with real robots. In this work, raw visual signals directly feed a neural network, and an actor-critic architecture is used to train the neural network. In these examples, the visual signal is downscaled and averaged into a monochrome (i.e. two-colors) image of $64 \times 24 = 1536$ pixels. The output of four infrared sensors are also added to this perceptual input. While the approach is effective for the specific tasks, this process can only be used in a highly controlled environment. Real-world images are much richer and could not undergo such a strong reduction in size. Trying to apply this approach to larger images would be fruitless because of the Bellman curse of dimensionality.

In a similar spirit, Ernst et al. have very recently shown the applicability of a novel reinforcement learning algorithm (*Fitted Q Iteration* [EGW05]) in conjunction with a powerful supervised learning algorithm (*Extra-Trees* [GEW06]) for closed-loop learning of visual policies when raw image pixels are directly used as the perceptual input [EMW06]. In their experimental setup, an image contains $30 \times 30 = 900$ grayscale pixels. This method is very close to the algorithm *Visual Approximate Policy Iteration* (V-API) that will be introduced in Chapter 6, but it has been developed independently and almost simultaneously [JBP06]. Contrarily to V-API, this experimental setup lacks an evaluation on real-world, full-sized images.

Another important, distinct step in the direction of my objectives was achieved by Piater et al., who have considered the aforementioned task of grasping objects by a dexterous manipulator [Pia01, CPG01, PG02]. Starting with the fact that only few pieces of work in robotics combine haptic and visual feedback for grasping [All84, GL86], Piater et al. propose to insert a visual recognition system in front of a grasp controller. This controller, which was initially designed by Coelho and Grupen [CJG97], is purely local and minimizes the wrench residual. The visual recognition system is aimed, given prior haptic experience, at planning an initial grasp that is close enough to the optimal grasp. This initial grasp will be optimized through the local controller. In other words, visual feedback ensures the convergence towards an optimum in the space of possible finger configurations. The mappings from images to successful finger configurations are learned through an online, incremental protocol [Pia01, Chapter 6]. This process generates discriminative combinations of basic visual features (edgels, texels and salient points), and each of these combinations votes for one configuration of the hand. A Bayesian network is trained to choose the best configuration from these votes. Unfortunately, as mentioned earlier, this algorithm is only applicable to tasks with two outcomes (the grasp is either successful or not) and in which two successive actions are independent. This application is therefore inherently closer to supervised learning than to reinforcement learning.

1.3.3 Extraction of Visual Features

The proposed algorithms resort to the *extraction of visual features* as a way to achieve more compact state spaces that can be used as an input to traditional RL algorithms. Indeed, buried in the noise and in the confusion of visual cues, images contain hints of regularity. Such regularities are captured by the important notion of *visual features*. Loosely speaking, a visual feature is a representation of some aspect of local appearance, e.g. a corner formed by two intensity edges, a spatially localized texture signature, or a color. Therefore, to analyze images, it is often sufficient for a computer program to extract only useful information from the visual signal, by focusing its attention on robust and highly informative patterns in the percepts. The program should thereafter seek the characteristic appearance of the observed scenes or objects.

This is actually the basic postulate behind *appearance-based vision* that has had much success in computer vision applications such as image matching, image retrieval and object recognition [SM97, Low04]. Appearance-based vision relies on the detection of stable discontinuities in the visual signal thanks to *interest point detectors* [SMB00, MTS$^+$05]. Similarities in images are thereafter identified using a *local description* of the neighborhood around the interest points [MS03, MS05]. If two images share a sufficient number of matching local descriptors, they are considered as belonging to the same visual class. Appearance-based vision is at the same time powerful and flexible as it is robust to partial occlusions and does not require segmentation or 3D models of the scenes. Mikolajczyk et al. provide a list of successful applications of appearance-based vision [MS05].

Intuitively, appearance-based vision allows the extraction of a maximum amount of relevant information from the images, while keeping the amount of data manageable. Interestingly enough, Paletta et al. have also exploited visual features in the context of reinforcement learning for optimizing sensorimotor behavior in the context of active vision [PRB05, PFS05].

1.3.4 Task-Driven Exploitation of Visual Features

It seems therefore promising to use the visual feature space instead of the raw images as the input to the reinforcement learning algorithms. Indeed, visual features are vectors that are composed of about a hundred components, which contrasts with the typical images that contain hundreds of thousands of pixels. Thus, reinforcement learning seems easier to carry out in this visual feature space. Nevertheless, many features can be extracted from a single image. In other words, the extraction of visual features from an image is a *one-to-many mapping*. Thus, the main challenge of the algorithms that will be developed is to determine which, among the many visual features it perceives, are the ones relevant to the reward or punishment.

In this dissertation, two ways to deal with this abundance of visual features are investigated. The first possibility is to select the visual features that are the most *discriminative for the performed task*. This basic idea has already been investigated by McCallum's *U Tree* algorithm [McC96]. This algorithm will be adapted to visual tasks in the *Reinforcement Learning of Visual Classes* algorithm. The name of the algorithm comes from the fact that the visual space is discretized into a small set of *visual classes* by testing the presence of those highly discriminative visual features.

The second possibility is to use the whole set of *raw* visual features, without selecting the most discriminative ones. This is motivated by the fact that even less discriminative features can have an impact on the optimal decisions. Instead of creating a set of visual classes, low-level vectors of real numbers that encode the visual features are directly used. As visual data can be sampled only sparsely, function approximators will be embedded inside the RL process. These ideas will lead to the development of the *Visual Approximate Policy Iteration* algorithm.

It is important to realize that the visual feature space can be composed of raw pixels. Indeed, the extraction of visual features can consist in the selection of informative patches in the image and in the description of each of those patches as the set of raw pixels they contain. Furthermore, the extraction of visual features can generate a single visual feature for each image. Therefore, the introduced framework can deal with the same perceptual spaces as those considered by Iida et al. [ISS02, SI03] and Ernst et al. [EGW05] (cf. Section 1.3.2).

1.4 Outline of the Dissertation

This dissertation targets the development of algorithms that make reinforcement learning computationally tractable on complex, high-dimensional visual tasks. In a nutshell, the basic idea is to take advantage, in an automatic way, of visual features that are borrowed from appearance-based vision. The results of this dissertation unify those appearing in previous publications [JP04, JP05a, JP05b, JP05c, JP05d, JP05e, JSP05, JBP06, JP06a, JP06b].

Each algorithm that will be introduced in the sequel comes from a practical consideration that was guided by the experiments. Ideas from experiments and programming influenced the theoretical development, and *vice versa*. This goal-driven approach turned out to be very fruitful. The development of software in the `C` and in the `C++` programming languages forms a major part of my work. The main developments were: (a) a core object-oriented library for reinforcement learning that supports vision-for-action tasks and that is used throughout the experiments; (b) a library for the learning of classification and regression trees, and notably Extra-Trees [GEW06]; (c) a library for distributed computing of bag-of-tasks on a cluster of computers; and (d) the data distribution of a database through the BitTorrent protocol [Coh03].

This dissertation is organized as depicted in Figure 1.1. Because the topic of this research work is at the crossroads of computer vision and machine learning, this dissertation begins with two background chapters that present the state of the art of the two main theoretical tools that will be used:

- In Chapter 2, the basic results from the theory of reinforcement learning are presented. The formalism that is used throughout this dissertation is introduced. Theoretical foundations of RL are discussed, as well as standard algorithms. This chapter makes no assumptions abouth the state and action spaces, except that they must both be finite. Visual state spaces are not considered in this part of the dissertation. People from the computer vision community might find this an useful introduction to the vision-for-action paradigm.

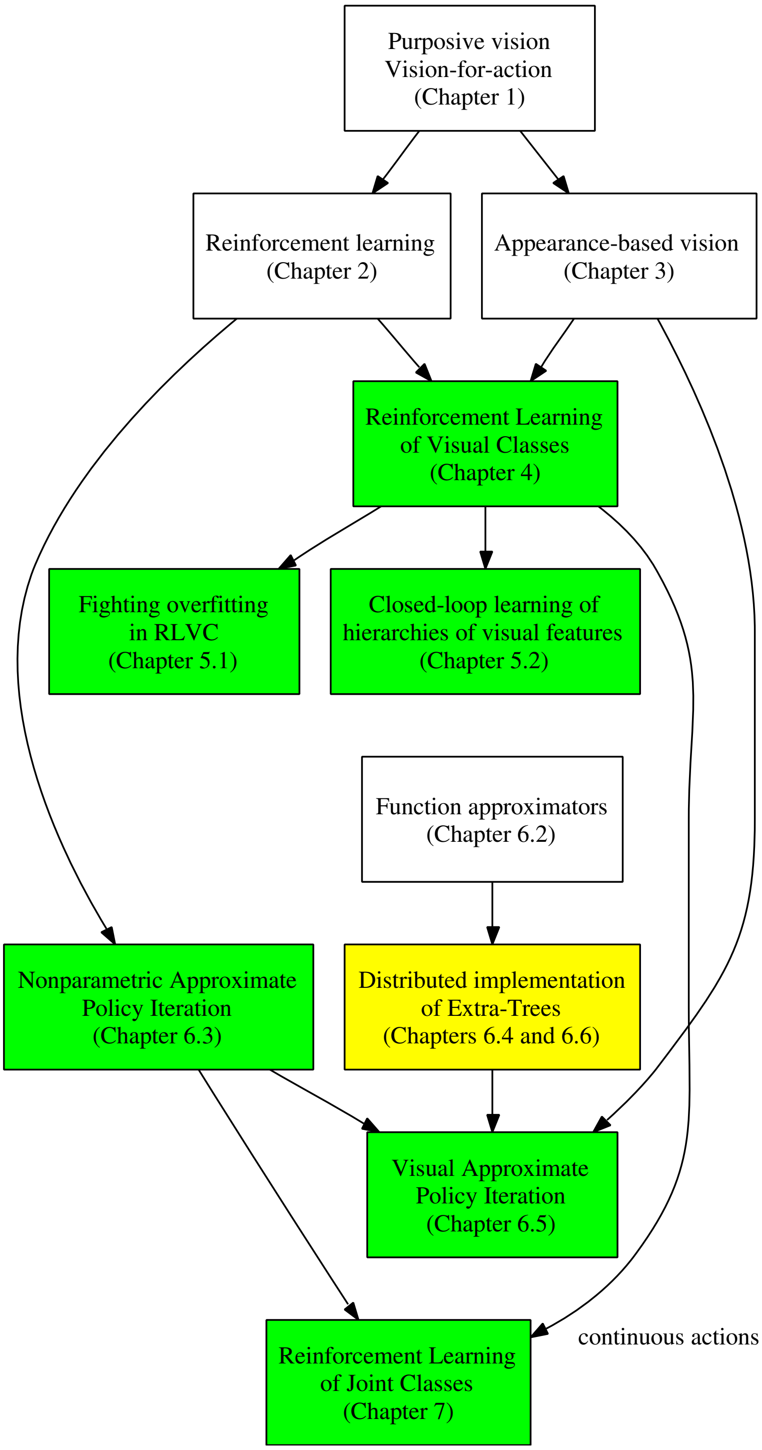


Figure 1.1: Synoptic view of this dissertation. Arrows represent the dependencies between the various concepts that are introduced in the chapters. My personal contributions are highlighted in green. The Extra-Trees learning algorithms result from research work by Geurts et al. [GEW06], but the distributed implementation of Extra-Trees is a personal contribution.

- Chapter 3 introduces appearance-based vision along with visual features. The central notion of a *visual feature generator* is introduced. Then, *global-appearance methods* and *local-appearance methods* are both discussed, the latter being defined as the conjunction of an *interest point detector* with a *local description generator*. This introductory chapter concludes by showing how visual features are used in typical computer vision applications. Some precise insight on how visual features could be used when solving vision-for-action tasks are also given. This discussion should be mostly helpful to people from the machine learning community.

After these two background chapters, the remaining chapters constitute the heart of this dissertation. These technical chapters share the same general structure: The precise problem and the proposed solution are stated, the related work is discussed, novel algorithms are formally derived, experimental results are presented, and a discussion concludes. The validity of the proposed methods are demonstrated on vision-guided navigation tasks. Here are the contributions that will be described:

- The Reinforcement Learning of Visual Classes (RLVC) algorithm is derived in Chapter 4 by building on the visual feature space defined in the preceding chapter. As mentioned earlier, this algorithm selects highly informative visual features in an incremental process.

- Chapter 5 presents two extensions to RLVC. The first extension consists in reducing the overfitting that is inherent to RLVC by resorting to techniques borrowed from computer-aided verification. Although this process is more resource consuming, it is useful to ensure better convergence properties. The second extension is closely related to Scalzo's research work [SP05, SP06], and proposes algorithms to generate a hierarchy of spatial combinations of visual features that are more and more discriminative. This process proves to be useful when individual features are not informative enough to solve a vision-for-action task. This is illustrated on a visual version of a classical control problem (the *car-on-the-hill* task).

- The Visual Approximate Policy Iteration (V-API) algorithm is introduced in Chapter 6. As mentioned earlier, this algorithm uses the raw visual features through a function approximation scheme. This chapter notably introduces the *Nonparametric Approximate Policy Iteration* algorithm, that is a generic version of the *Least-Squares Policy Iteration* algorithm [LP03]. The supervised learning of Extra-Trees [GEW06] is also described, as it is a main component of V-API. This chapter also shows how to distribute the generation of Extra-trees among a cluster of computers so as to dramatically reduce the computational requirements.

- All the algorithms from the previous chapters are defined on finite action spaces. Chapter 7 generalizes RLVC to continuous action spaces, leading to the more general *Reinforcement Learning of Joint Classes* (RLJC) algorithm.

> This is useful, as robotic controllers often interact with their environment through a set of continuously-valued actions (position, velocity, torque. . .). RLJC adaptively discretizes the joint space of visual percepts and continuous actions, which is in essence a novel approach.

Finally, Chapter 8 concludes this dissertation with a summary of the main contributions and with a discussion of possible future directions.

CHAPTER

TWO

Reinforcement Learning

Reinforcement Learning *(RL) is concerned with the closed-loop learning of a task within an* a priori *unknown environment. The learning agent takes lessons from a sequence of trial-and-error interactions with the surrounding environment. In RL, the task to be solved is not directly specified. Instead, whenever the agent takes a decision, it feels either pain or pleasure. This so-called* reinforcement signal *implicitly defines the task. The goal of the agent is to learn to act rationally, that is, to learn how to maximize its expected rewards over time. RL algorithms achieve this objective by constructing* control policies *that directly connect the percepts of the agent to the suitable reaction when facing these percepts.*

The agent is never told the best reaction when facing a given situation, neither whether it could reach better performance than the one it currently achieves. As a consequence, RL schematically lies between supervised learning and unsupervised learning. Indeed, in supervised learning, an external teacher always gives the correct reaction to the agent and the agent has to learn to reproduce the given input-output relation. On the other hand, the unsupervised learning protocol gives strictly no clue about the goodness of the decisions: The agent has to structure its percepts without getting feedback. The major advantages of the RL protocol are that it is fully automatic, and that it imposes only weak constraints on the environment.

As a consequence, RL can be distinguished from other learning paradigms by three main characteristics:

1. *the implicit definition of a goal through* reinforcements,
2. *the* temporal *aspect of the task (a decision can have a long-term impact, both on the system dynamics and on the earned rewards), and*
3. *the* trial-and-error *learning protocol (which contrasts with supervised and unsupervised learning).*

In this chapter, RL is introduced. Note that many textbooks present a more thorough coverage of the fields of reinforcement learning and its relation to the theory of dynamic programming [KLM96, BT96, SB98].

2.1 Markov Decision Processes

Reinforcement learning is traditionally defined in the framework of *Markov Decision Processes* (MDP). Bellman founded the theory of MDPs [Bel57a, Bel57b] by unifying previous work about sequential analysis [Wal47], statistical decision functions [Wal50], and two-person dynamic game models [Sha53].

The current section presents a self-contained introduction to finite Markov decision processes. The theorems that are useful in reinforcement learning for finite state-action spaces will be formally derived. The only statement that will not be proved is the Bellman optimality theorem (Theorem 2.15). Our aim is to emphasize how the main results about MDPs can be derived starting from the latter theorem.

The components that make up an MDP are rigorously defined in the subsequent sections. Notation that is similar, but not identical to that of Sutton and Barto [SB98], will be used.

2.1.1 Dynamics of the Environment

A Markov decision process is a stochastic control system whose state changes over time according to discrete-time dynamics, and whose evolution can be controlled by taking a sequence of decisions. Therefore, the trajectory that is followed by the system depends on the interactions between the "laws of motion" of the system and the decisions that are chosen over time.

At any time $t = 0, 1, \ldots$, the system can be observed and classified into one state s_t of a set of states S. At any time t, the learning agent influences its environment by taking one action a_t of a set of actions A, hereby controlling the system. The laws of motion of the MDPs are assumed to be governed by a time-invariant set of transition probabilities. The probability of reaching a state s_{t+1} after applying the action a_t in the state s_t does not depend on the entire history of the system, but only on the current s_t and a_t:

$$\mathrm{P}\left\{ s_{t+1} = s \mid \underbrace{s_0,\ a_0,\ s_1,\ a_1, \ldots,\ s_t,\ a_t}_{\text{history of the system}} \right\} = \mathrm{P}\{s_{t+1} = s \mid s_t,\ a_t\}. \tag{2.1}$$

This strong assumption on the system dynamics is generally known as the *Markov hypothesis*. Thus, the dynamics of MDPs can be entirely specified by defining a probabilistic relation $\mathcal{T}$ that links s_t, a_t and s_{t+1}:

$$\mathcal{T}(s, a, s') = \mathrm{P}\{s_{t+1} = s' \mid s_t = s,\ a_t = a\}. \tag{2.2}$$

Of course, the Markov hypothesis has many interesting implications that will be investigated in the next sections. Note however that despite the Markov hypothesis, an action can have a long-term impact on the trajectory of the system, and that the outcome of a decision is generally not perfectly predictable, because of the stochastic aspect of $\mathcal{T}$.

2.1.2 Reinforcement Signal

In MDPs, the control law that has to be learned is not directly specified. Rather, it is defined implicitly through a *reinforcement signal* that provides a quantitative evaluation of the reactions of the learning agent. Every time it takes a decision, the agent receives either a reward or a punishment, depending on its performance. So, the reinforcement signal tells *which* task is to be solved, but not *how* to solve the task. From a biological perspective, the reinforcement signal corresponds to pleasure or pain feelings that living beings may perceive while learning to achieve a task.

Concretely, at each time stamp t, the agent receives a real number $r_{t+1} = \mathcal{R}(s_t, a_t)$, that is called the reinforcement at time $t+1$ and that depends on the state s_t and on the action a_t that was performed in this state. In this framework, costs can be encoded as negative rewards.

Markov Decision Processes can now be formally defined as the assembly of a Markovian dynamics with a reinforcement signal:

Definition 2.1. A *Markov Decision Process* (MDP) is a quadruple $\langle S, A, \mathcal{T}, \mathcal{R} \rangle$, where:

- S is a set of *states*,
- A is a set of *actions*,
- $\mathcal{T} : S \times A \mapsto \Pi(S)$ is a probabilistic *transition relation* from the state-action pairs to the states, and
- $\mathcal{R} : S \times A \mapsto \mathbb{R}$ is the *reinforcement signal* that maps a state-action pair to a real number.

Remark 2.2. In this definition, the notation $R : E \mapsto \Pi(F)$ is employed to designate a probabilistic relation R that maps a set E to a set F. Thus, R is a probability density function over the Cartesian product $E \times F$. □

A very important subclass of Markov decision processes is constituted by those MDPs whose set S of states and set A of actions are both finite:

Definition 2.3. A *Finite Markov Decision Process* (FMDP) is a Markov decision process $\langle S, A, \mathcal{T}, \mathcal{R} \rangle$ such that S and A are finite sets.

2.1.3 Histories and Returns

As the agent interacts with its environment, the MDP describes a trajectory in the set of states. This trajectory depends on the actions that are chosen by the agent. So, the whole history of an MDP is a sequence of state-action pairs that is composed of the states that have been visited so far and of the previous actions that have been chosen when facing these states:

Definition 2.4. The *history* of an MDP up to time stamp $t \geq 0$ is a sequence $h_t = (s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t)$. The set $\mathcal{H}_t$ of possible histories up to t is:

$$\mathcal{H}_t = (S \times A)^t \times S, \tag{2.3}$$

and the set $\mathcal{H}$ of all possible histories is:

$$\mathcal{H} = \bigcup_{t \in \mathbb{N}} \mathcal{H}_t = \bigcup_{t \in \mathbb{N}} (S \times A)^t \times S. \tag{2.4}$$

To each history $h_t \in \mathcal{H}_t$ corresponds a sequence $r_0, r_1, \ldots, r_{t-1}$ of earned reinforcements, where $r_k = \mathcal{R}(s_k, a_k)$ for all $k < t$. Importantly, the immediate reward or punishment can be the consequence of decisions that were made long before. In other words, the reinforcements can be *delayed*. Therefore, the actions cannot be viewed independently of each other, and the agent may face a dilemma: Taking a less immediately attractive action can enable the agent to reach parts of the state space of the MDP where it can get higher future reinforcements. This makes for example particular sense when modeling two-person games such as chess, in which sacrificing a piece might lead to an advantage later in the game, or when modeling robotic tasks, in which each elapsed period of time induces a cost. Thus, the agent must balance its propensity for acquiring high immediate reinforcements with the possibility of earning higher rewards afterward.

Consequently, the concept of *returns* is introduced, that embodies this trade-off the learning agent has to make between present and future reinforcements. Given an infinite history $h \in \mathcal{H}_\infty$ of the interactions of the agent with the MDP, the corresponding return is the cumulative sum of reinforcements over time:

Definition 2.5. The *(discounted) return* $R(h)$ that is collected during an infinite history $h \in \mathcal{H}_\infty$ is:

$$R(h) = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t), \tag{2.5}$$

where $\gamma \in [0, 1[$ is the *discount factor*. Such a series always converges.

The temporal discount factor γ gives the current value of the future reinforcements. From a financial point of view, γ corresponds to the *time value money*, that is one of the basic concepts of finance: Money received today is more valuable than money received in the future by the amount of revenues it could yield. As it is assumed that $\gamma < 1$, the effect of distant future reinforcements becomes negligible. To intuitively paraphrase this definition, as future decisions influence the benefits of the current decision, the definition of discounted return stresses the short-term rewards, without totally neglecting the long-term consequences.

Note that as γ tends to 1, the agent takes long-term consequences more strongly into account. Conversely, if $\gamma = 0$, the agent is myopic and only tries to maximize its immediate reinforcements[1]

[1] By convention, $0^0 = 1$ and $0^t = 0$ if t is a non-zero positive integer.

2.1.4 Control Policies

The agent interacts with the MDP through its effectors by taking actions. Whenever the agent faces some state, it must choose a suitable action according to the history of the system. The internal process of choosing actions is captured by the notion of decision rule:

Definition 2.6. A *decision rule* $\delta : \mathcal{H} \mapsto \Pi(A)$ is a probabilistic mapping from the set of possible histories to the set of actions.

A decision rule $\delta(h_t, a_t)$ tells the agent the probability with which it should choose an action $a_t \in A$ if the history of the system is $h_t \in \mathcal{H}_t$ at a given time stamp t. To control the system over time, a sequence of such decision rules must be used, one for each time stamp:

Definition 2.7. A *general control policy* π is an infinite sequence of decision rules: $\pi = (\delta_0, \delta_1, \ldots, \delta_t, \ldots)$.

The primary objective of the theory of MDPs is to find general control policies that are optimal for a given MDP, in a sense that remains to be defined.

Three very important subclasses of general control policies are now discussed. Firstly, a general control policy is Markovian if the decision rules do not depend on the whole history of the system, but only on the current state s_t:

Definition 2.8. A general control policy $\pi = (\delta_0, \delta_1, \ldots, \delta_t \ldots)$ is *Markovian* (or *memoryless*) if, for each time stamp t, there exists a probabilistic mapping $\delta'_t : S \mapsto \Pi(A)$ such that $\delta_t(h) = \delta'_t(s_t)$ for all $h = (s_0, a_0, \ldots, s_t) \in \mathcal{H}_t$.

Markovian control policies are particularly attractive, as the agent is not required to memorize the entire history of its interactions with the environment to choose the dictated action. Intuitively, it seems natural to only consider Markovian control policies when solving MDPs because the dynamics of MDPs is itself Markovian. This insight will be confirmed later.

Secondly, if the same decision rule is used at each time stamp, the general control policy is called stationary:

Definition 2.9. A general control policy $\pi = (\delta_0, \delta_1, \ldots, \delta_t, \ldots)$ is *stationary* (or *time invariant*) if $\delta_i = \delta_j$ for all $i, j \in \mathbb{N}$.

Evidently, any decision rule δ can be extended to a control policy $\pi = (\delta, \delta, \ldots, \delta, \ldots)$.

Finally, if each decision rule defines a single-valued transform from the states to the actions, the general control policy is called deterministic:

Definition 2.10. A general control policy $\pi = (\delta_0, \delta_1, \ldots, \delta_t \ldots)$ is *deterministic* if, for each time stamp t and each possible history $h_t \in \mathcal{H}_t$, there exists one action a_t such that $\delta_t(h_t, a_t) = 1$.

As a shorthand, if π is deterministic, the notation $\delta_t(h_t)$ will refer to the action that is selected with probability 1 by the decision rule δ_t if faced with the history $h_t \in \mathcal{H}_t$.

Remark 2.11. If a policy π is at the same time Markovian and stationary, the policy collapses to a probabilistic mapping $S \mapsto \Pi(A)$ from the states to the actions. In such case, $\pi(s, a)$ will designate the probability of choosing action $a \in A$ when facing some state $s \in S$. If π is moreover deterministic, $\pi(s)$ will refer to the action that is selected with probability 1. □

2.1.5 Value Functions

Suppose that, starting in a particular state, actions are taken following a fixed control policy. Then the expected sum of rewards over time is called the value function of the policy that is followed. More precisely, each general control policy π is associated with a value function $V^\pi(s)$, that gives for each state $s \in S$ the expected discounted return obtained when starting from state s and thereafter following the policy π:

Definition 2.12. The *value function* $V^\pi : S \mapsto \mathbb{R}$ of a general control policy π for any state $s \in S$ is:

$$V^\pi(s) = \mathrm{E}^\pi \left\{ R(h) \mid s_0 = s \right\} = \mathrm{E}^\pi \left\{ \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \mid s_0 = s \right\}, \tag{2.6}$$

where E^π denotes the expected value if the agent follows π, starting with an history h_0 that only contains s. $V^\pi(s)$ is called the *utility* or the *value* of the state s under the policy π.

Evidently, the value function for a given general control policy is by definition unique. It is now proved that value functions of Markovian, stationary control policies satisfy a very specific recursive relation in finite MDPs. This property expresses a relationship between the value of a state and the values of its successor states:

Theorem 2.13 (Bellman equation). Let π be a Markovian, stationary control policy for a finite MDP $\langle S, A, \mathcal{T}, \mathcal{R} \rangle$. Using the notation from Remark 2.11, we get:

$$V^\pi(s) = \sum_{a \in A} \pi(s, a) \left(\mathcal{R}(s, a) + \gamma \sum_{s' \in S} \mathcal{T}(s, a, s') V^\pi(s') \right), \text{ for each } s \in S. \tag{2.7}$$

Proof. The proof is given in Appendix A. □

If the considered policy is also deterministic, this theorem can be readily specialized as:

Corollary 2.14. If π is a Markovian, stationary, *deterministic* control policy in a finite MDP, then:

$$V^\pi(s) = \mathcal{R}(s, \pi(s)) + \gamma \sum_{s' \in S} \mathcal{T}(s, \pi(s), s') V^\pi(s'), \text{ for all } s \in S. \tag{2.8}$$

2.2 Dynamic Programming

Any Markov decision process induces a problem of sequential decision making and of stochastic control, that will be referred to as a *Markov decision problem*. *Dynamic Programming* (DP) was introduced by Bellman [Bel57a] as a way to solve various recursive problems that happen to include Markov decision problems. We now give some important results from the theory of dynamic programming and investigate the link between DP algorithms and MDPs.

Note that all the elements from the previous sections (system dynamics, returns, control policies, value functions) can be defined even when the state space S and the action space A are infinite. However, from this point on, the study will be restricted to finite MDPs. A great deal of literature is devoted to such models [How60, Der70, Whi93, Tij94, Ber95, Ber00]. Most of the following results also hold when considering infinite state spaces and/or infinite action spaces, if some continuity conditions on $\mathcal{T}$ and $\mathcal{R}$ are met [Hin70, HLL96, HLL99].

We conclude this introduction by mentioning work about continuous-time dynamics in the MDP literature. A noticeable extension of MDPs is that of *continuous-time MDPs* [How60], in which the system evolution is described continuously in time, and in which rewards are accumulated continuously in time. Such models can be converted into discrete-time models through a discretization process that is nowadays known as *uniformization* [Lip75, Ser79]. It is also possible to define *Semi-Markov Decision Processes* [How71] that relax the assumption of actions with exponential delay distributions that is embedded inside the definition of continuous-time MDPs.

2.2.1 Markov Decision Problems

Value functions lie at the heart of the theory of DP. Indeed, once a state $s \in S$ is fixed, they allow to rank the policies according to their utility for this state. DP is interested in a more general problem, that is to compute policies that maximize the utility of *all* the states of the MDP. Obviously, proving the existence of such optimal policies is not immediate. This is precisely the topic of the following central theorem that is attributed to Bellman:

Theorem 2.15 (Bellman optimality theorem). There exists at least one Markovian, stationary, deterministic control policy π^* such that:

$$V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s), \text{ for all } s \in S. \tag{2.9}$$

All general policies that satisfy this relation are called *optimal policies*. Furthermore, all optimal policies share the same unique value function, that is denoted $V^*(s)$ and that satisfies:

$$V^*(s) = \max_{a \in A} \left(\mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') V^*(s') \right), \text{ for all } s \in S. \tag{2.10}$$

V^* will be referred to as the *optimal value function*.

Proof. Derman provides a rigorous treatment of this result [Der70, Theorem 3.1]. Note that it is possible to prove that any general control policy can be converted into an equivalent Markovian policy (i.e. that has the same value function), independently of any optimality consideration [Whi93]. □

Intuitively, Equation 2.10 expresses the fact that the utility of a state under an optimal policy is equal to the expected return for taking the best action from that state.

This theorem not only claims the existence of an optimal *general* control policy, but also that it is always possible to find an optimal *Markovian, stationary, deterministic* control policy. This result is pretty remarkable, as it implies that optimal decisions can always be taken without taking care of time, neither of the history of the system. The possibility of focusing on such a very simple class of policies is mostly due to the Markovian assumption that is embedded inside the definition of MDPs. This explains why introductory work about MDPs and reinforcement learning generally focus on such a particular class of policies. This also motivates the following definitions:

Definition 2.16. A *percept-to-action mapping* $\pi : S \mapsto A$ is defined as a Markovian, stationary, deterministic control policy. For such policies, the shorthand $\pi(s)$ will designate the action that is deterministically selected by the control policy at any time stamp when facing the state $s \in S$.

Definition 2.17. Given an MDP $\langle S, A, \mathcal{T}, \mathcal{R}\rangle$ and a discount factor γ, the *Markov decision problem* is defined as the computation of a percept-to-action mapping that is optimal with respect to the MDP.

2.2.2 Contraction Mappings

Markov decision problems have just been formalized. Classical DP algorithms that enable the solving of MDPs given the full knowledge of its structure $\langle S, A, \mathcal{T}, \mathcal{R}\rangle$ are now described. For this purpose, it is important to notice that Equations 2.7 and 2.10 share a similar recursive structure: They relate the utility of a state to that of the successors of this state. Let π be a Markovian, stationary control policy. Two convenient transforms (resp. called T^π and T) are now introduced, that map a value function V to another value function (resp. $T^\pi V$ and TV):

$$(T^\pi V)(s) = \sum_{a\in A} \pi(s,a) \left(\mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') V(s') \right), \text{ and} \tag{2.11}$$

$$(TV)(s) = \max_{a\in A} \left(\mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') V(s') \right), \tag{2.12}$$

for all $s \in S$. Using these transforms, Equations 2.7 and 2.10 can be respectively rewritten under a more compact form:

$$T^\pi V^\pi = V^\pi, \text{ and} \tag{2.13}$$

$$TV^* = V^*. \tag{2.14}$$

The mapping T is often referred to as the *Bellman backup operator* in the literature.

T^π and T should not be understood as just a shorthand notation. Indeed, both T^π and T are contraction mappings, and such contraction mappings have many interesting properties in the framework of dynamic programming.

Definition 2.18. Let $||\cdot||$ be a norm on $\mathbb{R}^n$ and γ be a scalar in $[0,1[$. Let $C : \mathbb{R}^n \mapsto \mathbb{R}^n$ be a mapping from vectors of real numbers to vectors of real numbers. C is a *contraction mapping* if there exists a constant $\rho \in [0,1[$ such that:

$$||C\boldsymbol{x} - C\boldsymbol{y}|| \leq \rho||\boldsymbol{x} - \boldsymbol{y}||, \tag{2.15}$$

for all $x, y \in \mathbb{R}^n$. ρ is called the *contraction factor* of C.

Theorem 2.19. T^π and T are contraction mappings with respect to the maximum norm $||V||_\infty = \max_{s\in S}|V(s)|$.[2]

Proof. The proof is postponed to Appendix A. □

Contraction mappings play a crucial role in DP thanks to the following general result:

Theorem 2.20 (Banach Fixed Point Theorem). Let $C : \mathbb{R}^n \mapsto \mathbb{R}^n$ be a contraction mapping. Then, (a) there exists a *unique* point $\boldsymbol{x}^* \in X$ such that $C(\boldsymbol{x}^*) = \boldsymbol{x}^*$ (i.e. C admits a unique fixed point), and (b) for every $\boldsymbol{x}_0 \in X$, the sequence:

$$\boldsymbol{x}_0, C(\boldsymbol{x}_0), C^2(\boldsymbol{x}_0), \ldots, C^t(\boldsymbol{x}_0), \ldots$$

converges to $\boldsymbol{x}^*$ geometrically. In particular, the following holds for any $t \in \mathbb{N}$:

$$||C^t(\boldsymbol{x}_0) - \boldsymbol{x}^*|| \leq \rho^t||\boldsymbol{x}_0 - \boldsymbol{x}^*||. \tag{2.16}$$

Proof. Bertsekas et al. provide a comprehensive proof of this result [BT89]. □

Note that the Banach fixed point theorem immediately implies the uniqueness of the optimal value function, which was already stated in Theorem 2.15. It will also be used to prove the convergence of DP algorithms. Furthermore, it allows the use of a simple procedure for extracting an optimal percept-to-action mapping π^* from the optimal value function V^*:

Theorem 2.21. Let π^* be a percept-to-action mapping defined as[3]:

$$\pi^*(s) = \operatorname*{argmax}_{a\in A}\left(\mathcal{R}(s,a) + \gamma\sum_{s'\in S}\mathcal{T}(s,a,s')V^*(s')\right), \tag{2.17}$$

for all $s \in S$. Then, π^* is an optimal policy.

[2] Here, we take some liberty with the notation. Strictly speaking, a value function is not a vector of real numbers, and thus cannot be a contraction mapping as defined earlier. However, as S is finite, each state $s \in S$ can be uniquely associated with an index in $\{1, \ldots, |S|\}$. Therefore, there exists a bijection between value functions for a state space S, and vector of real numbers of $\mathbb{R}^{|S|}$. So, we can talk indifferently about value functions and vectors of reals.

[3] Note that π^* can be assumed deterministic: If more than one action reaches the maximum for some state $s \in S$, then π^* chooses one arbitrary, fixed action among this set.

Proof. The proof uses Banach fixed point theorem, see Appendix A. □

Intuitively, this result shows that, once the optimal value function V^* has been computed, an optimal percept-to-action mapping π^* can be built taking a greedy action with respect to V^*. Therefore, the problem of computing an optimal policy for an MDP is equivalent to the problem of computing the optimal value function. Indeed, if an optimal policy π^* is given, it is possible to compute V^* through Equation 2.7. Conversely, an optimal policy π^* can be extracted from V^* through Equation 2.17.

This fundamental result is at the basis of two well-known DP algorithms, *Value Iteration* and *Policy Iteration*, that are described in the next sections. In Value Iteration, the optimal value function V^* is first computed, then an optimal percept-to-action mapping π^* is extracted from V^*, whereas in Policy Iteration, an optimal percept-to-action mapping π^* is directly computed. A third general approach exists, but it will not be discussed further in this dissertation. It consists in formulating the Markov decision problem as a linear programming problem [Man60]. The interested reader can found more detail in Derman's book [Der70], as well as in some surveys [KLM96, SB98].

2.2.3 State-Action Value Functions

Before describing Value and Policy Iteration, a last important concept in the study of Markov decision problems is introduced. We remark that extracting an optimal policy π^* through Equation 2.17 requires the knowledge of both $\mathcal{R}$ and $\mathcal{T}$, even if V^* is known. To put it in other words, the optimal value function does not embed the stochastic aspect of the MDPs, but the knowledge of this aspect is necessary to choose the best reaction. Clearly, this is a limitation of value functions, which becomes a true liability in reinforcement learning. Indeed, RL algorithms do not assume the knowledge of the underlying MDP, which contrasts with DP algorithms.

This motivates the introduction of state-action value functions[4], that are a convenient way to embed, in a single framework, the dynamics of the environment and the value functions. State-action value functions became an important part of reinforcement learning with the introduction of *Q-learning* (cf. Section 2.4.2) by Watkins [Wat89, WD92]. Whereas in reinforcement learning state-action value functions are standard, they are not often considered explicitly in the literature about Markov decision processes and dynamic programming.

Given a general control policy π, the state-action value function gives the expected discounted return obtained by starting from some state $s \in S$, taking some action $a \in A$, and thereafter following π:

Definition 2.22. The *state-action value function* $Q^\pi : S \times A \mapsto \mathbb{R}$ of a general

[4] In the RL literature, state-action value functions are often referred to as *Q-functions* or *quality functions*. Sutton provides a discussion of this terminology [Sut04] and why it should be avoided.

control policy π for any state $s \in S$ and any action $a \in A$ is:

$$Q^{\pi}(s,a) = \mathrm{E}^{\pi}\left\{\sum_{t=0}^{\infty} \gamma^{t} \mathcal{R}(s_t, a_t) \mid s_0 = s, a_0 = a\right\}. \tag{2.18}$$

As a direct consequence of the definitions, the following equations relate the value function and the state-action value function of a Markovian, stationary policy π:

$$\begin{aligned} V^{\pi}(s) &= \sum_{a \in A} \pi(s,a) Q^{\pi}(s,a), \text{ and} && (2.19)\\ Q^{\pi}(s,a) &= \mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') V^{\pi}(s'), && (2.20)\end{aligned}$$

for all $s \in S$ and $a \in A$. Likewise, the Bellman equation 2.13 can be adapted as:

$$Q^{\pi}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') \sum_{a' \in A} \pi(s',a') Q^{\pi}(s',a'). \tag{2.21}$$

The counterpart of Bellman optimality theorem for state-action value functions states that all the optimal policies share the same unique state-action value function, that is denoted $Q^{*}(s,a)$ and that satisfies:

$$\begin{aligned} Q^{*}(s,a) &= \max_{\pi} Q^{\pi}(s,a), \text{ and} && (2.22)\\ Q^{*}(s,a) &= \mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') \max_{a' \in A} Q^{*}(s',a'). && (2.23)\end{aligned}$$

Q^{*} will be referred to as the *optimal state-action value function*. Finally, once the optimal state-action value function Q^{*} is known, it is possible to extract an optimal percept-to-action mapping π^{*} along with the optimal value function V^{*}:

$$\begin{aligned} \pi^{*}(s) &= \operatorname*{argmax}_{a \in A} Q^{*}(s,a), \text{ and} && (2.24)\\ V^{*}(s) &= \max_{a \in A} Q^{*}(s,a), && (2.25)\end{aligned}$$

for all $s \in S$. Indeed, Theorem 2.21 has shown that an optimal percept-to-action mapping π^{*} takes a greedy action with respect to Q^{*}. Very importantly, these two equations do not require the knowledge of the transition relation $\mathcal{T}$. This contrasts with the optimal value function V^{*}.

We conclude this section by introducing two additional Bellman backup operators, denoted H^{π} and H, that map a state-action value function to another state-action value function. These transforms play the same role as T^{π} and T for value functions. Given a Markovian, stationary control policy π, they are defined as:

$$\begin{aligned} (H^{\pi}Q)(s,a) &= \mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') \sum_{a' \in A} \pi(s',a') Q(s',a'), \text{ and} && (2.26)\\ (HQ)(s,a) &= \mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') \max_{a' \in A} Q(s',a'), && (2.27)\end{aligned}$$

for all $s \in S$ and $a \in A$. Similarly than for value functions, Bellman equation and Bellman optimal equation can be shortened as:

$$\begin{aligned} H^\pi Q^\pi &= Q^\pi, \text{ and} \qquad (2.28) \\ HQ^* &= Q^*. \qquad (2.29) \end{aligned}$$

2.2.4 Value Iteration

Value Iteration is a DP algorithm that is due to Bellman [Bel57a]. It computes the optimal state-action value function Q^* by constructing a sequence of state-action value functions $Q_0, Q_1, \ldots, Q_i, \ldots$ through the following iterative rule:

$$Q_{i+1} = HQ_i, \qquad (2.30)$$

starting with an arbitrary state-action value function Q_0 (generally, $Q_0(s) = 0$ for all states $s \in S$). A stopping criterion that is commonly used for this update rule is to terminate when $||Q_{i+1} - Q_i||_\infty$ drops below a fixed threshold ε. Convergence of this process to Q^* is guaranteed thanks to the Banach fixed point theorem and to the fact that H is a contraction mapping (cf. Theorem A.3 of Appendix A). An optimal percept-to-action mapping π^* is thereafter deduced by taking the greedy action with respect to Q^* through Equation 2.24.

A more algorithmic description of Value Iteration was given by Sutton and Barto [SB98]. The version of Value Iteration we have presented is Jacobi-like: Two separate state-action value functions Q_{i+1} and Q_i are used. It is worth pointing out that a Gauss-Seidel variant of Value Iteration exists. In this setup, the updated components of equation $Q_{i+1} = HQ_i$ are used as soon as they become available. The latter version also converges. For a discussion of asynchronous value iteration in general we refer to Bertsekas and Tsitsiklis [BT96].

Remark 2.23. Value Iteration is generally presented as an iterative algorithm that computes the optimal value function V^* by applying the update rule $V_{i+1} = TV_i$. As discussed in Section 2.2.2, this process also converges, because T is a contraction mapping. This differs from the version of Value Iteration we have just presented that targets the optimal *state-action* value function Q^*. This choice is mostly conventional: It will smooth the transition between dynamic programming and reinforcement learning. Furthermore, it will facilitate the description of the *Visual Approximate Policy Iteration* algorithm in Chapter 6. □

2.2.5 Policy Iteration

The Policy Iteration algorithm is a second important DP algorithm that is usually attributed to Howard [How60]. Rather than learning the V^* or the Q^* optimal functions as in Value Iteration, Policy Iteration directly learns an optimal control policy. Starting with an initial, arbitrary percept-to-action mapping π_0, Policy Iteration constructs successive improved policies $\pi_1, \pi_2, \ldots$ by relying on two interleaved learning components that are described below: Policy Evaluation and Policy Improvement.

Policy Evaluation

The Policy Evaluation component computes the state-action value function $Q^{\pi_k}(s,a)$ of the current policy π_k. A possible way of computing $Q^{\pi_k}(s,a)$ is to solve Equations 2.7 for obtaining $V^{\pi_k}(s)$ then to extract $Q^{\pi_k}(s,a)$ through Equation 2.20, or, equivalently, to directly solve Equations 2.21. These are indeed simple systems of respectively $|S|$ and $|S| \times |A|$ linear equations that can be solved using an algorithm as simple as Gauss-Seidel elimination. An in-depth discussion can be found in Howard [How60, pp. 34–37,81–83].

The main weakness of this approach is that the dimension of the system depends on the number of states, which causes performance issues when the state space is large. This problem can be overcome to some extent if distributed computation among a cluster of computers is used [BT89]. However, a more commonly used approach for Policy Evaluation is to deal with this problem in a way that is very similar to the Value Iteration algorithm. Indeed, the H^{π_k} mapping is also a contraction mapping (cf. Theorem A.4 of Appendix A), so an iterative algorithm that is almost identical to Value Iteration can be used. The only difference with Value Iteration is that H is replaced by H^{π_k} in the update rule. Concretely, this version of Policy Evaluation starts with a Q_0 state-action value function that is taken equal to $Q^{\pi_{k-1}}$, then generates a sequence of state-action value functions $Q_1, \ldots, Q_i, \ldots$ through the following iterative rule:

$$Q_{i+1} = H^{\pi_k} Q_i. \tag{2.31}$$

The algorithm stops when the difference between Q_{i+1} and Q_i in the maximum norm drops below a threshold. This process is often referred to as *Modified Policy Evaluation* [PS78]. One must be aware that Modified Policy Evaluation generally leads to an approximation of Q^{π_k}, whereas solving of the system of linear equations produces the exact result. The insight is that an *exactly* evaluated policy is not required in order to improve it. The choice of $Q_0 = Q^{\pi_{k-1}}$ as a starting point reduces the number of iterations before convergence, because the policy π_k generally shares common decisions with π_{k-1}.

Policy Improvement

The Policy Improvement step consists in improving the current policy π_k by choosing a strictly improving action in as many states as possible. This step uses the Q^{π_k} that was computed by the Policy Evaluation step to generate an improved policy π_{k+1}. This improved policy π_{k+1} is usually chosen so as to be the greedy policy with respect to Q^{π_k}, i.e. so as to maximize the state-action value function Q^{π_k}:

$$\pi_{k+1}(s) = \operatorname*{argmax}_{a \in A} Q^{\pi_k}(s,a), \tag{2.32}$$

for all $s \in S$. The soundness of this approach is proved by showing that the value function of π_{k+1} is everywhere at least as good as the value function of π_k:

Theorem 2.24 (Policy Improvement theorem). Let π and $\tilde{\pi}$ be two Markovian, stationary control policies. Assume that $\tilde{\pi}$ is such that:

$$\sum_{a \in A} \tilde{\pi}(s,a) Q^{\pi}(s,a) \geq V^{\pi}(s), \text{ for all } s \in S. \tag{2.33}$$

Then, we have:

$$V^{\tilde{\pi}}(s) \geq V^{\pi}(s), \text{ for all } s \in S. \tag{2.34}$$

Proof. An intuitive proof of this theorem is provided in Sutton and Barto [SB98]. An in-depth discussion is provided in Appendix A. □

Obviously, if π_{k+1} is defined as in Equation 2.32, the hypothesis of this theorem is verified. Thus, we conclude that π_{k+1} is at least as good as π_k in every state with respect to their value function.

Discussion

Once the policy has been improved, we evaluate the new policy and again try to improve it. The algorithm stops when there are no strictly improving action, i.e. when $\max_{a \in A} Q^{\pi_k}(s,a) = V^{\pi_k}(s)$ for every state $s \in S$. Indeed in this case, Equation 2.20 gives:

$$V^{\pi_k}(s) = \max_{a \in A} Q^{\pi_k}(s,a) = \max_{a \in A} \left(\mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') V^{\pi_k}(s') \right). \tag{2.35}$$

Consequently, the Bellman optimality theorem 2.15 states that the value function V^{π_k} of π_k corresponds to the optimal value function V^*. We conclude that π_k is an optimal control policy.

As mentioned earlier, Policy Estimation together with Policy Improvement give rise to the Policy Iteration algorithm. Once again, the interested reader can find an algorithmic treatment of Policy Iteration in Sutton and Barto [SB98]. If Modified Policy Evaluation is used instead of solving a linear system of equations, the resulting algorithm is known as Modified Policy Iteration [PS78]. Note that Value Iteration can be thought of as the extreme case of Modified Policy Iteration when the Modified Policy Evaluation step carries out only a single update.

Importantly, because S and A are both assumed to be finite, the number of possible percept-to-action mappings is also finite. Therefore, Policy Iteration always converges in a finite number of iterations, if the Policy Evaluation step is achieved by resolving a system of linear equations. Furthermore, the number of improvements that are needed before convergence is typically small because each Policy Improvement step is based on accurate information about the value function of the current policy [How60, BT89]. This makes of Policy Iteration an attractive DP algorithm, if the computational complexity that is intrinsic to the Policy Evaluation process is neglected.

We conclude this section by pointing out that Policy Iteration can be interpreted as an instance of the actor-critic paradigm [BSA83, KT03]. Actor-critic methods are made up of two complementary parts: the *critic* that evaluates the current performance of the learning agent, and the *actor* that is responsible for the optimization of the current policy with respect to the conclusions of the critic. Such methods are discussed in more detail in Sutton and Barton [SB98, Section 6.6]. In the case of Policy Iteration, the critic component corresponds to Policy Evaluation, whereas the actor component consists in the Policy Improvement mechanism.

2.3 Generic Framework of Reinforcement Learning

We now build on the theory of MDPs to define that of reinforcement learning. Similarly to dynamic programming algorithms, reinforcement learning algorithms target the solution of Markov decision problems. However, the main difference to DP is that RL does not assume the *a priori* knowledge of the structure of the MDP that has to be solved. A RL agent is aware of the state space S it can perceive through its sensors, of the action space A it can span through its effectors, but *not* of the system dynamics $\mathcal{T}$, *nor* of the reinforcement signal $\mathcal{R}$. This lack of a model requires sampling the MDP to gather sufficient statistical knowledge about this unknown model: The characteristics of the environment can only be determined by performing actions in the environment and observing the results. As a consequence, exploration is crucial for the RL process to succeed.

In RL, the agent strives to learn an optimal control policy by trial and error from its interactions with the surrounding Markov decision process. Schematically, any RL algorithm operates by repeating the following general sequence of operations: At time t,

1. It senses its inputs to determine the current state s_t of the environment;

2. It selects an action a_t according to its control policy at time t;

3. It applies this action, which results in sensing a new state s_{t+1} while perceiving a numerical reinforcement $r_{t+1} \in \mathbb{R}$; and

4. It updates its knowledge about the surrounding MDP and/or it modifies its current control policy.

According to this general scheme, a single execution step of an RL algorithm can be characterized by four elements: the initial state s_t, the chosen action a_t, the perceived reinforcement r_{t+1} and the resulting state s_{t+1}. This leads to the definition of an interaction:

Definition 2.25. An *interaction* is a quadruple:

$$\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle \in S \times A \times \mathbb{R} \times S. \tag{2.36}$$

Therefore, RL is defined as the solution of Markov decision problems given a finite sequence of interactions with the environment.

Note that at the second step of the RL scheme, an action a_t is selected. Importantly, the way a_t is chosen can be totally unrelated to the optimal control policy that is to be learned. Sutton and Barto talk about the "behavior policy" when designating the policy that is used at step 2 [SB98]. The behavior policy is to be distinguished from the "estimation policy" that is iteratively constructed and improved by the RL algorithm, and that will eventually lead to the optimal control policy. The estimation policy is often not directly represented. Indeed, many RL algorithms learn an estimation of the optimal state-action value function and the estimation policy is implicitly defined as the greedy policy with respect to this function (cf. Equation 2.24).

Some RL algorithms allow the behavior policy to be fully randomized: The agent keeps acting randomly, seemingly not taking immediate lessons from its interactions with environment, but updating its internal model of the estimation policy. After a certain amount of such random interactions, the algorithm stops and returns the estimation policy, which has hopefully converged to the optimal control policy. Such methods are often referred to as *off-policy* (or *exploration insensitive*) methods. On the other hand, the estimation policy can be used directly as the behavior policy. It this case, the behavior policy is continuously improved after each interaction, visibly converging to an optimal control policy. This leads to the so-called *on-policy* (or *exploration sensitive*) methods.

Although on-policy methods can seem much more spectacular from the AI point of view, the development of off-policy methods is one of the biggest successes of RL. Indeed, off-policy methods are able to learn from a static sequence of interactions that has been independently collected. This makes them more flexible and less dangerous, as it is not required to directly connect the learning agent (that initially mostly act randomly) to the real-world system for the learning to succeed.

2.4 Reinforcement Learning in Finite Domains

To summarize, RL algorithms learn an optimal control policy π^* from a finite database of interactions. Of course, the generic RL scheme we have just presented can be instantiated as a RL algorithm in various ways. We now review several instances of the RL paradigm that can deal with finite state-action spaces. We particularly focus the discussion on two well-known instances of the RL paradigm: model-based methods and Q-learning. These algorithms will be useful in the sequel.

However, it is important to point out that a large part of the RL literature is devoted to infinite domains. Indeed, we note that the generic framework of reinforcement learning does not assume the finiteness of the state space, neither of the action space. Recent work in reinforcement learning generally focuses on infinite state spaces. The discussion of such algorithms is deferred to Chapter 6. Similarly, RL algorithms that can deal with infinite action spaces are discussed in Chapter 7. We also mention that the RL paradigm can be adapted to continuous time by considering a continuous formulation of the Bellman equations, that are known as Hamilton-Jacobi-Bellman equations [Doy96, MBM99, Doy00, Cou03, Mun06b].

2.4.1 Model-Based Algorithms

The most direct way for solving the reinforcement learning problem is to reconstruct the surrounding MDP by estimating the reinforcement signal $\mathcal{R}$ and the transition relation $\mathcal{T}$ from the database of interactions. Once this estimation is built, dynamic programming algorithms such as Value Iteration or Policy Iteration can be applied. This way, the RL problem is reduced to that of DP. Such approaches are often referred to as *model-based* methods.

Extracting $\mathcal{T}$ from a finite database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$ (with $t = 1, \ldots, N$) can for example be achieved by computing the relative frequencies that appear in the database. Similarly, $\mathcal{R}$ can be estimated as the mean of the perceived reinforcements for each state-action pair. Formally, consider two states $s, s' \in S$, an action $a \in A$ and a time stamp $t \in \{1, \ldots, N\}$. We define the following shorthands:

$$\eta_t(s,a) = \begin{cases} 1 & \text{if } s_t = s \text{ and } a_t = a, \\ 0 & \text{otherwise;} \end{cases} \tag{2.37}$$

$$\xi_t(s,a,s') = \begin{cases} 1 & \text{if } s_t = s, a_t = a \text{ and } s_{t+1} = s', \\ 0 & \text{otherwise;} \end{cases} \tag{2.38}$$

$$\zeta(s,a) = \sum_{t=1}^{N} \eta_t(s,a). \tag{2.39}$$

$\zeta(s,a)$ corresponds to the number of interactions that have s as starting point and a as selected action. Using this notation, the estimated transition relation and reinforcement signals can be defined as:

$$\mathcal{T}(s,a,s') = \begin{cases} \sum_{t=1}^{N} \frac{\xi_t(s,a,s')}{\zeta(s,a)} & \text{if } \zeta(s,a) \neq 0, \\ 1 & \text{if } \zeta(s,a) = 0 \text{ and } s' = s, \\ 0 & \text{otherwise;} \end{cases} \tag{2.40}$$

$$\mathcal{R}(s,a) = \begin{cases} \sum_{t=1}^{N} \frac{r_t \eta_t(s,a)}{\zeta(s,a)} & \text{if } \zeta(s,a) \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.41}$$

The values for state-action pairs such that $\zeta(s,a) = 0$ are purely conventional and reflect a lack of exploration of the environment. Other prior estimate could be used: For instance, in Equation 2.40, one might assume that all transitions are equiprobable, thus setting $\mathcal{T}(s,a,s') = 1/|S|$ whenever $\zeta(s,a) = 0$.

Of course, model-based methods are inherently limited to batch processing: The RL process can only start when the database of interactions has been collected. As a consequence, they are off-policy methods. In practice, as they make very effective use of available data, model-based algorithms give very good results, on the strict condition that the number of state-action pairs is relatively small and that the database is representative enough. If this is not the case, function approximation techniques are needed, which will be introduced later.

2.4.2 Q-Learning

Q-learning is certainly the most well-known model-free RL algorithm [Wat89]. This algorithm directly learns the optimal state-action value function Q^*. It was inspired by the *temporal difference* learning process that was previously proposed by Sutton [Sut88]. It is popular both for its convergence results and for its ease of implementation.

Q-learning stepwise updates an estimation of Q^* through a *bootstrapping* process: The estimate of some state-action optimal value is updated with the estimated optimal value of a successor state. Recall the Bellman optimality equation:

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,s') \max_{a' \in A} Q^*(s',a'). \tag{2.42}$$

Suppose that $\hat{Q}$ is a state-action value function that corresponds to the current estimation of Q^*. Let now $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ be an interaction. Let also $(s,a) \in S \times A$ be a state-action pair. Then the value $\max_{a' \in A} \hat{Q}(s',a')$ is an estimation of $\sum_{s' \in S} \mathcal{T}(s,a,s') \max_{a' \in A} \hat{Q}(s',a')$ if the successor s' is chosen with probability $\mathcal{T}(s,a,s')$. But following the transitions of the environment ensures making a transition from s to s' with probability $\mathcal{T}(s,a,s')$. Thus

$$r_{t+1} + \gamma \max_{a' \in A} \hat{Q}(s_{t+1}, a') \tag{2.43}$$

is an unbiased estimate of the optimal state-action value $Q^*(s_t, a_t)$ that is derived from the considered interaction (cf. Equation 2.42). The *temporal difference* at time t is the difference between this observed estimate and the old estimate:

$$\Delta_t = r_{t+1} + \gamma \max_{a' \in A} \hat{Q}(s_{t+1}, a') - \hat{Q}(s_t, a_t). \tag{2.44}$$

This value Δ_t is often equally referred to as the *Bellman residual* at time t. Note that if the system is deterministic, the Bellman residuals are equal to zero as soon as $\hat{Q}$ has converged to the optimal state-action value function by virtue of the Bellman optimality theorem. Based on this temporal difference, we define a new estimate $\hat{Q}'$ of the optimal state-action value function as:

$$\hat{Q}'(s,a) = \begin{cases} \hat{Q}(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t, \\ \hat{Q}(s,a) + \alpha \Delta_t & \text{otherwise,} \end{cases} \tag{2.45}$$

where α is the *learning rate* parameter that balances the importance of the bootstrapping. Thanks to this parameter, each state-action value $\hat{Q}(s,a)$ progressively converges to the expected future rewards in the face of stochastic transitions of the environment.

Following the steps detailed in this discussion, Q-learning builds a sequence of state-action values functions $\hat{Q}_0, \hat{Q}_1, \ldots$ by repeatedly using the following update rule whenever a new interaction $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ is acquired:

$$\hat{Q}_{t+1}(s_t, a_t) = \hat{Q}_t(s_t, a_t) + \alpha_t(s_t, a_t) \left(r_{t+1} + \gamma \max_{a' \in A} \hat{Q}_t(s_{t+1}, a') - \hat{Q}_t(s_t, a_t) \right), \tag{2.46}$$

leaving all the state-action pairs that are different from (s_t, a_t) unchanged. The step-size parameter $\alpha_t(s_t, a_t)$ may change from one iteration to the next and may depend both on s_t and a_t. Note that Q-learning is an off-policy method, thus the behavior policy can be distinct from the estimation policy.

Convergence Properties

The convergence of Q-learning has been investigated by several authors [WD92, JJS94, Tsi94, BT96]. It has been proved that this algorithm converges with probability 1 to the optimal state-action value function if the behavior policy visits each state-action pair infinitely often and if the learning rate α_t decreases to zero at a suitable rate. Given an infinite sequence of interactions, the following conditions precisely define what is a “suitable rate”:

$$\sum_{t=0}^{\infty} \alpha_t(s,a) \quad = \quad \infty \tag{2.47}$$

$$\sum_{t=0}^{\infty} \alpha_t^2(s,a) \quad < \quad \infty, \tag{2.48}$$

for each state-action pair $(s, a) \in S \times A$. For practical purposes, a common choice for α_t that meets these two requirements is:

$$\alpha_t(s_t, a_t) = \frac{1}{1 + v(s_t, a_t)}, \tag{2.49}$$

where $v(s_t, a_t)$ denotes the number of visits to the state-action pair (s_t, a_t) before time t. This choice requires the storage of a counter for each state-action pair.

The proof of convergence of Bertsekas and Tsitsiklis relies on the fact that the optimal state-action function Q^* is a fixed point of the Bellman backup operator H [Tsi94, BT96]. Another approach is to relate Q-learning to the framework of stochastic approximation theory that was defined by Robbins and Monro [RM51], as Q-learning can be described as a stochastic version of Value Iteration (compare Equations 2.42 and 2.43) [JJS94].

Exploration vs. Exploitation Trade-Off

As mentioned earlier, Q-learning is an off-policy algorithm. However, contrarily to model-based methods, it is not limited to batch processing and progressively learns while interacting with the environment: Q-learning is an *on-line* method, whereas model-based approaches are *off-line* methods [KLM96]. It is therefore possible for Q-learning to control the system at the same time it learns an optimal policy.

In such an on-line setup, Q-learning must not only randomly *explore* the system to gather more accurate knowledge about its dynamics, but should also *exploit* what it has learned so far by greedily choosing actions with respect to its current estimate of the Q^* function. If there is insufficient exploration, the algorithm will likely converge to a sub-optimal control policy. On the other hand, deferring exploitation tends to disregard the on-line control of the system. As a consequence, there exists a trade-off between exploration and exploitation. Evidently, this trade-off is not unique to Q-learning and must be faced by any on-line RL algorithm. This has led to the development of *exploration strategies* that choose the actions by balancing exploration with exploitation. In essence, an exploration strategy tells how to derive a behavior policy from an estimation policy.

Let $\hat{Q}(s,a)$ be the current estimate of Q^* of a fixed RL algorithm at a given time. The *ε-greedy* (or *semi-uniform random*) exploration strategy [Wat89] is the simplest one. This strategy depends on a parameter parameter $\varepsilon \in [0,1]$. The behavior policy it defines either picks an action at random with probability ε, or follows the greedy action $\text{argmax}_{a\in A} \hat{Q}(s,a)$ with probability $1-\varepsilon$. Formally, the ε-greedy strategy defines a Markovian, stationary, non-deterministic behavior policy $\pi : S \mapsto \Pi(A)$ (cf. Remark 2.11) as:

$$\pi(s,a) = \begin{cases} \frac{1-\varepsilon}{M} & \text{if } \hat{Q}(s,a) = \max_{a'\in A} \hat{Q}(s,a'), \\ \frac{\varepsilon}{|A|} & \text{otherwise,} \end{cases} \tag{2.50}$$

where M is the number of actions $a \in A$ such that $\hat{Q}(s,a) = \max_{a'\in A} \hat{Q}(s,a')$.

The *Boltzmann* (or *Gibbs*, or *softmax*) exploration strategy [Luc59, Bri90] is slightly more sophisticated and is based on the Boltzmann distribution. Instead of assigning the same probability to each action $a \in A$ when a random selection is performed in a state $s \in S$, the Boltzmann strategy weights the actions according to the value $\hat{Q}(s,a)$: Actions leading to higher expected returns are more likely to be selected. A *temperature parameter* $T \geq 0$ determines the amount to which the probabilities are affected. A zero temperature leads to a fully greedy behavior policy, whereas a temperature that tends to infinity generates a uniform random policy. Thus, varying the temperature defines a smooth continuum between greedy and random policies. The behavior policy $\pi : S \mapsto \Pi(A)$ that is induced by the Boltzmann strategy is:

$$\pi(s,a) = \frac{e^{\hat{Q}(s,a)/T}}{\sum_{a'\in A} e^{\hat{Q}(s,a')/T}}. \tag{2.51}$$

These two exploration strategies satisfy the convergence conditions of Q-learning. In practice, the ε parameter or the temperature T is gradually reduced, in order to progressively move from a purely exploratory policy to an exploitation policy as the RL algorithm better captures the dynamics of the environment.

Both of these so-called *undirected strategies* have limitations. Indeed, the proper initial values for ε and T, as well as their decaying rate, are problem specific and must often be hand-tuned: They have no way to detect when enough exploration has taken place. This can be penalizing when interactions are expensive to acquire. More complex exploration strategies, referred to as *directed strategies*, solve this problem to some extent by keeping track of which areas of the underlying MDP have been explored so far [RP03]. This is generally done by counting the number of visits to each state-action pair (s,a) and by storing information about the statistical confidence in the values $\hat{Q}(s,a)$.

Directed strategies comprise *interval estimation* [Kae90, KLM96] and *model-based interval estimation* [Wie99]. Very sketchily, these methods compute a confidence interval for each state-action value, then choose the action with the highest upper bound. Initial exploration is assured because of an initially large confidence interval, and the exploration decreases as the confidence in the estimated optimal state-action values grows. The *counter-based* [Thr92], the *recency-based* [Thr92], and the *error-based* [Sch91] directed exploration strategies are also commonly used.

2.4.3 Survey of Other Algorithms

Of course, model-based methods and Q-learning are just a small subset of the RL algorithms that have been proposed over years in the literature. We now give some pointers to other algorithms that are applicable when dealing with finite state-action spaces. Additional material can be found in reference textbooks [KLM96, BT96, SB98].

Temporal-Difference Algorithms

Q-learning belongs to the family of *temporal-difference methods* (cf. Section 2.4.2). *SARSA* is another algorithm from this family [RN94]. This method uses an update rule that is syntactically similar to that of Q-learning, but that is conceptually quite different. Unlike Q-learning, SARSA is an on-policy algorithm: Instead of learning the optimal state-action value function, it learns the state-action value function *of the behavior policy* and continuously improves this policy. SARSA gets its name from its update rule that learns from a sequence of quintuplets $\langle s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}\rangle$ [Sut96b]. Each such quintuplet consists in an interaction along with the action a_{t+1} that is chosen by π at the next time step, when faced with state s_{t+1}. Then, the update rule is:

$$\hat{Q}_{t+1}(s_t, a_t) = \hat{Q}_t(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_{t+1} + \gamma\underline{\hat{Q}_t(s_{t+1}, a_{t+1})} - \hat{Q}_t(s_t, a_t)\right), \tag{2.52}$$

where the underlined part of the formula is the only difference with respect to the update rule of Q-learning (cf. Equation 2.46). Theoretical analysis shows that SARSA also converges [SJLS00].

Adaptive Heuristic Critic (AHC) [BSA83] is a third temporal-difference method that is a stochastic version of Policy Iteration (just as Q-learning can be thought of as the stochastic version of Value Iteration). AHC is an on-policy algorithm that is defined in the actor-critic framework: The critic component evaluates the value of its behavior policy π through a stochastic version of Policy Estimation, and the actor component regularly improves the behavior policy. In general, the critic component is implemented through the TD(0) algorithm [Sut88], or through an evolution of this algorithm. TD(0) updates its estimate of the value function $V^\pi(s)$ of the behavior policy π through the stochastic version of the Bellman equation (cf. Theorem 2.13):

$$\hat{V}_{t+1}(s_t) = \hat{V}_t(s_t) + \alpha_t(s_t, a_t)\left(r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)\right), \tag{2.53}$$

whenever an interaction $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$ is acquired by following π.

In AHC, the behavior policy π is not extracted from a value function as in Q-learning or SARSA, but it is stored independently according to a suitable representation scheme. Moreover, contrarily to Policy Iteration, AHC does not assume any knowledge of the dynamics of the environment and, as such, cannot directly use Equation 2.32 to improve the behavior policy. Therefore, several versions of AHC have been proposed that mostly differ in the way the behavior policy π is stored and in the way the actor improves its policy according to the temporal-difference that is computed by the critic component. One possible variant consists in storing a set of *preferences* $p(s, a)$ of selecting an action a when facing a state s, then in increasing (resp. decreasing) this preference whenever the temporal-difference of Equation 2.53 is positive (resp. negative) [SB98, Section 6.6].

Although AHC has been exploited in many early RL systems, its use tends to decline in favor of methods based on state-action value functions such as Q-learning or SARSA [SB98, Section 6.6]. However, the AHC model of reinforcement learning is still appealing from a biological point of view [Bar95, HAB95, KLG$^+$05].

Eligibility Traces

The learning rules of Q-learning, SARSA and TD(0) only take the current state into account when updating their estimate of the (state-action) value functions. However, the immediate reinforcement r_{t+1} that is perceived in the interaction at time t also influences to a lesser extent the return that is perceived at time $t-1$ (cf. Equation 2.5). Similarly, r_{t+1} has an even lighter impact on the return at time $t-2$, and so on. This suggests using a more general, elaborate procedure for updating the value functions: Instead of updating only the utility of the state s_t, the perceived reinforcement could be backpropagated over the utility of the previous states along the trajectory that has been followed by the MDP. The backpropagation is carried out with exponentially decaying weights that are governed by a decay parameter $\lambda \in [0, 1]$, hence capturing the discounting that is embedded in the definition of the return. This way, convergence is hopefully quicker.

This basic idea has been unified under the name of *eligibility traces*. Conceptually, an eligibility trace is a function $e_t : S \mapsto \mathbb{R}$ that records to what extent an immediate reinforcement has an impact on each state of the MDP. The stochastic update rule (cf. Equations 2.46, 2.52 and 2.53) is then applied to *all* the states $s \in S$, and not only to the current state s_t, weighting the temporal-difference update by the value $e_t(s)$ of the eligibility trace. The eligibility traces may be seen as an algebraic trick by which rewards are propagated backward over the current trajectory without having to remember the trajectory explicitly. The literature generally distinguishes two mechanisms for updating eligibility traces between two successive iterations: *accumulating traces* and *replacing traces* [SB98, Section 7.8]. Examples of algorithms that take advantage of eligibility traces are $Q(\lambda)$-learning [Wat89, PW96][5],

[5] These papers present two different versions of $Q(\lambda)$-learning: In Watkins' approach, the eligibility traces are reset whenever exploration takes place, which contrasts with Peng and Williams'

SARSA(λ) [RN94, Rum95], TD(λ) [Sut88, Day92], and, more recently, QV(λ)-learning [Wie05]. Singh and Sutton provide an in-depth theoretical discussion about eligibility traces [SS96].

Monte-Carlo Methods

Another approach to model-free reinforcement learning bases the prediction of the value functions on *average returns* collected while interacting with the environment. Such methods are known as *Monte-Carlo methods* [Rub81, BD94].

For example, let π be a percept-to-action mapping whose state-action value function $Q^\pi(s, a)$ has to be computed. Sketchily, the Monte-Carlo approach generates a large number of truncated histories that start with the state-action pair (s, a), and evaluates $Q^\pi(s, a)$ as the average of the observed returns corresponding to these histories. By virtue of the law of large numbers, this average value indeed converges to the true state-action utility. Thus, an optimal control policy can be generated through the Policy Iteration process: The state-action value function $Q^{\pi_k}(s, a)$ for the current policy π_k is estimated, then π_k is improved through Equation 2.32, and this process is repeated until convergence [SB98, Chapter 5].

Interactive Model-Based Techniques

The model-based algorithms that were presented in Section 2.4.1 are purely batch processing: Learning can only happen when the database of interactions has been fully collected. This might be problematic, as fully randomized behavior policies can be harmful for the surrounding environment, or can be very inefficient in gathering sufficient data when some states are hard to reach [Whi91][6]. The *interactive model-based* methods (also known as *planning* [SB98, Chapter 9]) relax this assumption and are capable of on-line learning. They can indeed produce an estimation of an optimal control policy *before* the end of the collection of the database.

The simplest interactive model-based method is the *certainty equivalence* approach [KV86]. Whenever M interactions are acquired, a model of the underlying MDP is estimated and solved through a dynamic programming algorithm (cf. Section 2.4.1). Unfortunately, this straightforward approach involves re-solving an MDP after each sequence of M interactions, hereby introducing a computational burden. This problem can be somewhat circumvented by taking the lastly computed optimal state-action value function as a starting point of the DP algorithm. However, there is in general no guarantee that this will indeed reduce the computation time.

Alternatively, instead of applying the Bellman backup operator H to *all* the states as in Value Iteration (cf. Section 2.2.4), H could be applied only to the state-action pairs whose utility are likely to change the most, given a change in the model.

version. These are probably the two most common implementations, though several other variants have been proposed [Lin93, Cic95, WS98].

[6] Note that the latter problem disappears if the environment supports resetting to a random state, which will be the case in the experiments of the next chapters, or if one uses a behavior policy that counts the number of visits to each state-action pair.

This idea has led to the development of *Dyna* [Sut90], *prioritized sweeping* [MA93] and *Queue-Dyna* [PW93]. These algorithms essentially differ in the way the state-action pairs to be updated are selected: In Dyna, the selection is random, whereas prioritized sweeping and Queue-Dyna make use of specialized data structures for deciding which transitions have the greatest impact on the optimal state-action value functions. Ernst provides an independent treatment of a prioritized sweeping-like algorithm [Ern03, Section 5.3]. *Real-time dynamic programming* [BBS95] is another interactive model-based algorithms that is tuned for specific problems [KLM96].

Finally, the interactive model-based algorithm that is called *explicit explore or exploit* (E^3) [KS98] has attracted a lot of attention since its introduction. Its main advantage is to explicitly balance exploration and exploitation, hereby achieving near-optimal performance in polynomial time. To this end, E^3 keeps track of the accuracy of its model of the environment. Recent work tries to combine E^3 with apprenticeship to eliminate its need for an initial aggressive exploration of the state space, which might cause problem in real-world tasks [AN05].

2.5 Successful Applications

We now list some of the most successful applications of reinforcement learning on real-world tasks. Of course, this is not an attempt to be exhaustive. A complete description of the used algorithms cannot be included in this dissertation, so we only provide pointers to the pertaining papers:

- Turning a computer into an excellent Backgammon player [Tes95];
- Turning a computer into an average chess player [BTW98];
- Dispatching elevators for skyscrapers [CB98];
- Learning quadruped gait control [HG98, KYK01, KS04b];
- Solving the Acrobot swing-up control problem [SB98, YIS99];
- Riding a bicycle [RA98, LP03];
- Training robotic agent for the RoboCup competition [SS01];
- Applications in trading [MS01b] and marketing [AVAS04];
- Controlling a helicopter [BS01, NCD+04];
- Stabilizing electric power systems [Ern03];
- Learning ball acquisition on an Aibo robot [FS04];
- Learning strategies for curing HIV [ESGW06];...

Although they are often limited, these applications are wide-ranging, targeting games, robotics, or even finance. Interestingly enough, psychology, biology and neuroscience that initiated the development of reinforcement learning, have also begun to benefit from the recent developments in the RL field [ODS+04, KLG+05].

2.6 Summary

This chapter introduced the theory of Markov decision processes and of reinforcement learning. In this framework, the learning agent iteratively discovers a percept-to-action mapping that maximizes the quantitative feedback it receives over time. The relation between reinforcement learning and dynamic programming was discussed. Several reinforcement learning algorithms for finite state-action spaces were presented, notably model-based methods and Q-learning.

CHAPTER

THREE

Appearance-Based Vision

As motivated in the Introduction, reinforcement learning is an especially attractive framework to deal with purposive vision problems. When RL is used in this context, the set of states S corresponds to a set of images. Unfortunately, it is impossible to directly feed basic RL algorithms with a state space that consists of images. Because such algorithms generally rely on a tabular representation of the value functions, they indeed become quickly impractical as the number of possible percepts increases. This is evidently a problem in visual tasks: Even though the set of possible images is finite, the number of possible images exponentially increases with their size, and images are highly subject to noise.

Similar problems often arise in many fields of computer vision, and several approaches have been proposed over years. Recent solutions often rely on the extraction of meaningful visual features *that can then be used for higher-level treatment. More specifically, we now discuss the popular, highly successful* local-appearance paradigm. *In the subsequent chapters, we will introduce novel RL algorithms that take advantage of the local-appearance paradigm, enabling them to solve purposive vision problems.*

3.1 Mid-Level Representation of Images

As argued above, images are high-dimensional and noisy. However, only a few portions of an image may contain contain relevant information for the task to be solved. Furthermore, typical images exhibit many redundant patterns. Therefore, a successful approach in computer vision systems consists in extracting *visual features* that digest the informative content of the raw images. Visual features lead to a mid-level representation of the images, that is halfway between the raw images and a more semantic representation of the observed objects. This approach can also be motivated from a neurophysiological point of view: Although the exact natural process is still currently debated, the human brain performs a data reduction of several order of magnitude to summarize the huge quantity of raw visual information it perceives, so that this “purified” data stream can be used in higher-level cognitive tasks.

3.1.1 Visual Feature Generators

Generally speaking, a visual feature is a vector of real numbers that represents some essential, discriminant and meaningful portion of an image. Visual features have a much smaller dimensionality than the image space, targeting a compact representation that improves and speeds up the image analysis. Furthermore, though it is not strictly required, visual features often add some invariance to viewpoint or illumination changes, with respect to the raw pixels of the original image. From an abstract point of view, the extraction of the visual features in an image is done by a visual feature generator:

Definition 3.1. A *visual feature generator* $\mathcal{G}_V : S \mapsto \mathcal{P}(V)$ is a mapping from the set of images S to the power set of visual features. V is the *visual feature space*[1], which usually corresponds to $\mathbb{R}^n$ for some $n > 0$.

A visual feature generator transforms an input image as a set of visual features. For computational tractability, it is also required that it be impossible to generate an infinite set of visual features: For each image $s \in S$, $\mathcal{G}_V(s)$ must be finite.

The various visual feature generators can be ranked according to their fulfillment of desirable properties, notably:

- *distinctiveness* (generated visual features should discriminate between different scenes or objects, so as to provide strong consistency constraints for matching),
- *robustness* (features should be weakly influenced by a bad image quality due to noise, blur, discretization artifacts or compression effects),
- *repeatability* (features should be invariant to scale, rotational, viewpoint or illumination changes),
- *simplicity* (features should be quickly extracted), and
- *compactness* (features should be low-dimensional, so as to speed up the matching process and to reduce the memory burden).

Evidently, these properties must be balanced with respect to the task that has to be solved. For example, a real-time robotic application typically requires both high simplicity and compactness. On the other hand, an object recognition task needs high distinctiveness, which can generally only be achieved by using a less compact representation. At the time of writing, there is no visual feature that performs uniformly better than the others.

[1] This definition allows the use of general visual feature spaces. In particular, visual features can be vectors of real numbers with varying size: $V = \cup_{i \in I} \mathbb{R}^i$, with $I \subseteq \mathbb{N}_0$. This possibility will not be exploited in this dissertation, but opens the path to the multimodal analysis of visual features. This might be useful when solving real-world robotic applications, because it allows the simultaneous use of different classes of visual features, combining their respective skills. Lisin et al. have recently proposed an example of such a multimodal analysis for image classification [LMB+05].

Remark 3.2. From a fully general perspective, the generation of visual features is stochastic. This means that the resulting set of visual features is possibly different from one application of $\mathcal{G}_V$ to the other, so that $\mathcal{G}_V$ is a probabilistic mapping $S \mapsto \Pi(\mathcal{P}(V))$. However, stress is generally put on deterministic visual feature generators. A noticeable exception is the work by Marée et al. [MGPW05] that will be discussed later. □

3.1.2 Appearance-Based Vision

Visual features have attracted a lot of attention in the literature over the recent years, leading to the very successful *appearance-based* approach to computer vision.

This trend must be distinguished from the older *geometry-based vision* (or *model-based vision*) that detects geometric primitives such as lines, curved surfaces or generalized cylinders in the images, and that maps them in a 3D model of the objects through algorithms such as RANSAC [FB81]. The target applications were localization and pose estimation from a single view of an object. Forsyth and Ponce's textbook provides a survey of these methods [FP03, Chapter 18]. Geometry-based vision is less flexible than appearance-based vision, because it requires the *a priori* knowledge of the geometry of the objects. Furthermore, the appearance of the objects (e.g. color and texture) is not fully taken into account. Therefore, geometry-based vision is currently mostly useful in tasks that consider simple, untextured manufactured objects.

Structure-based vision is another trend in computer vision. It analyzes the topology of visual structures, and is closely related to mathematical morphology [Ser82]. In such approaches, the topology of the observed objects is represented as a graph, and the matching of images is reduced to graph matching. The *skeleton* is an important notion in structure-based vision [Blu67]. It converts a binary image (i.e. a silhouette) to a graph that represents the articulations connecting the maximal balls that can be embodied inside the silhouette. The skeleton can be extracted from the *contours* that are lines along which an abrupt change in the luminosity occurs [Can86]. Theoretical advantages of the skeleton include its invariance to linear transformations (scale, rotation, translation), and the fact that it preserves topological and geometrical properties of the silhouettes. Unfortunately, it is highly sensitive to noise and to an inaccurate binarization of the images. Moreover, similarly to geometry-based vision, it only considers binary images and thus does not bring the appearance of the objects into play. In practice, structure-based vision is mostly useful for applications in controlled environments such as biology or mineralogy. It has also been applied to image compression and architecture. Nowadays, structure-based vision tends to focus on the analysis of *ridges*, that do not require a binarization of the images and that are more informative about the shape of the objects [Ebe96].

Geometry-based and structure-based vision will not be further covered in this dissertation. Several instantiations of visual features and their application in computer vision are now discussed. Of course, a complete, exhaustive survey of appearance-based vision is out of the scope of this dissertation. Our aim is only to give some insight about state-of-the-art techniques and to present the most popular visual features.

3.2 Global-Appearance Methods

Historically, the visual features that have first been considered describe the *global appearance* of the image. Many visual structures, such as texture, are indeed only meaningful when the images are looked at in a large area. Global-appearance features digest the entire set of the pixels of the images. Corresponding visual feature generators have the property that a single visual feature is extracted, whatever the input image is: For each image $s \in S$, we have that $|\mathcal{G}_V(s)| = 1$.

Sometimes, depending on the targeted application, global appearance features are only computed on a *region of interest* in the image. In the case of video sequences, the region of interest can be delimited through tracking (e.g. using Kalman filtering [Kal60]) or motion segmentation (e.g. using background subtraction with a probabilistic background model that consists of a mixture of Gaussians [SG99]). On the other hand, the problem of segmenting static images is at best hard and not clearly defined, at worst an utopia: Segmenting an image should indeed extract meaningful regions of interest, but these are precisely determined by a high-level understanding of the image content, which generally leads to a chicken-and-egg situation. Nevertheless, the BlobWorld system is a successful application of segmentation to image retrieval [CBGM02]. Standard algorithms for image segmentation are thresholding, *Region Growing* and *Split and Merge* [PP93]. More advanced techniques notably include *Expectation-Maximization* [DLR77, BCGM98], and graph-based approaches such as the *Normalized Cut* algorithm [SM00] or its more general version, namely the *Graph Cut* [BVZ01].

3.2.1 Normalized Images

The most basic global-appearance feature generator is the *normalized image extractor*. Given an image $s \in S$, it is downsized to a fixed-size image and possibly converted to grayscale. Photometric normalization can also be carried out. All the n pixels in this normalized image are then collected inside a vector of real numbers to form a visual feature $\boldsymbol{v} \in \mathbb{R}^n$, thus the visual feature space is $V = \mathbb{R}^n$. The resulting visual feature can be matched with a model feature by directly resorting to a suitable correlation measure that is defined over the $\mathbb{R}^n$ space, such as the *Sum of Squared Distances* [CM95].

This approach has several weaknesses: Such features have a low robustness and repeatability, and are highly redundant; their distinctiveness is very dependent on the correlation measure; the memory and computational requirements are high if the model database contains various objects under various viewpoints; and insignificant variations in the images (such as minor scale changes) are susceptible to compromise the matching. This is mostly due to the non-exploitation of the knowledge about the distribution of the model features inside $\mathbb{R}^n$. However, normalized images have reported to be successful when they are directly used as feature vectors for *Support Vector Machines* (SVM) classification [PV98].

3.2.2 Eigen-Patches

To cope with these problems, Sirovich and Kirby have proposed to project the high-dimensional normalized images into a low-dimensional subspace [SK87]. Given a database of model images, *Principal Component Analysis* (PCA) is used to extract the first N eigen-patches that best explain the variations in the model images (N is chosen much smaller than n). Mathematically, the eigen-patches correspond to the first eigen-vectors of the covariance matrix of the model database. Any normalized image can thereafter be projected as a vector $\boldsymbol{v} \in \mathbb{R}^N$ of real numbers, and can be approximately reconstructed as the linear combination of the eigen-patches with coefficients given by $\boldsymbol{v}$. This process imposes some form of structure on the normalized images. In this framework, the visual feature space V corresponds to $\mathbb{R}^N$. Successful applications of PCA or variants include face description [SK87], face recognition [TP91], tracking [NMN94], object recognition [MN95], image retrieval [SW96] and visual navigation [WC96, WSV99].

3.2.3 Histograms

Rather than using PCA, it has been proposed to collect statistics about the image inside a histogram. *Histogram-based* approaches consider a set of M real-valued measures that can be evaluated at each pixel of the image. The variation domain of each measure is quantized into a set of buckets. Given an input image, the corresponding multidimensional histogram is computed as a vector of real numbers that counts, for each bucket of each measure, the proportion of pixels in the image the measure of which falls into this bucket. If m designates the total number of buckets across all the measures, the visual feature space is $V = \mathbb{R}^m$. Note that in general, it is difficult to determine the optimal bucket size for an application. Histogram-based approaches have become very popular for image retrieval.

Color Histograms

The simplest measure for histograms consists, given a pixel, in returning the value of a fixed color channel in a fixed colorspace. This kind of measure leads to the definition of *color histograms* [SB91], that are generally 3-dimensional. The seminal work about color histograms considers the widespread RGB colorspace [SB91]. By construction, this approach is robust to scale and rotational transforms. Unfortunately, color histograms based on the RGB colorspace are highly sensitive to the illumination conditions and are device-dependent. Furthermore, RGB is not perceptually uniform, which leads to unexploited and redundant bins if using a uniform quantization. As a consequence, the normalization of color histograms has been investigated [JV96], as well as colorspaces that offer improved uniformity such as HSV [SC95], CIELub [STLC97] or CIELab [CTB+99].

Texture Histograms

Because color histograms do not keep track of the relative arrangements of colored pixels, they are highly sensitive to pose and viewpoint changes. Therefore, *texture histograms* (also known as *texton histograms*) that take the texture of the observed objects into consideration have been proposed. The texture can indeed be a distinctive information for several classes of objects (think for example about grass, wood beams or brick walls).

There is no formal definition of what a texture is, but it intuitively corresponds to a spatial arrangement of small patterns. These patterns can repeat, though they are often fractal or random. In practice, a texture measure evaluates the response of a linear filter at the considered pixel. Schiele and Crowley consider the first-order Gaussian derivatives and the Laplacian as linear filters [SC96, SC00]. Leibe and Schiele measure Gaussian derivatives at multiple scales [LS03]. Renninger and Malik use filter banks giving answers that are similar to those given by the V1 visual cortex [RM04]. Recent advances in texture histograms compensate the shadowing and occlusions effects that are due to local height variations [OD04]. In general, texture histograms have a much higher dimensionality than color histograms. Note that histograms combining color and texture information have been explored [NBE+93].

3.3 Local-Appearance Methods

Because they consider all the pixels of the images, global-appearance methods suffer from two major drawbacks: They are not robust to partial occlusions, and they are very sensitive to background clutter. Removing clutter requires segmentation, but the segmentation of a scene is very costly, if not impossible to carry out in practice if a video sequence is not available. These limitations are also present in geometry-based and structure-based vision.

For this reason, vision systems that work by characterizing the entire image have been gradually supplanted over the last decade by *local-appearance methods*. Such methods identify statistically or structurally significant portions of the images, then represent each selected image portion as a vector of real numbers that is known as a *local descriptor*. The problem of matching images is consequently reduced to that of matching the local descriptors. Advantages of local-appearance methods are summarized below:

- There is no need for 2D or 3D models of objects[2];
- Local-appearance is robust to partial occlusions (and also to specular effects);
- No segmentation is required;

[2] Note that global-appearance methods have the same advantage.

- The local descriptors can compensate to some extent the local geometric and photometric deformations that result from seeing a scene under different viewing conditions, because projective deformations can be locally approximated as affine deformations, and because the local descriptors are generally normalized;

- The visual feature extraction is a biologically plausible process, which has been corroborated through neurophysiological experiments on the V1 visual cortex [KvD87, You87].

Local-appearance methods have been used in a broad class of computer vision problems, such as image classification and categorization, object detection and recognition, texture recognition, database retrieval, camera calibration, stereovision, structure from motion, robot localization and tracking [MS05].

In practice, local-appearance methods involve two elements: The *interest point detector* selects the most informative and reliable portions of the image, and the *local descriptor generator* converts the selected portions to a vector of real numbers that characterize the appearance of these portions. The portions selected by the interest point detector are centered around a so-called *interest point* in the image, and the portion itself is called the *neighborhood* of the interest point. In general, the interest points have very stable properties, such as a maximum in the signal entropy or an important change of the signal energy in all the directions. Thus, the selected neighborhoods are hopefully a direct result of a structure that is present in the scene, and there is therefore a high confidence that these neighborhoods will appear in many different views of a scene. Local-appearance vision has benefited in recent years from more and more powerful detectors and descriptors.

3.3.1 Harris Corner Detector

In this section, we provide an introduction to the most popular, important interest point detector, namely the *Harris detector* (that is also referred to as *Harris-Stephens detector*) [HS88]. We detail its principle, as it is the basis of most modern interest point detectors. It considers grayscale images and was originally inspired by the *Moravec corner detector* [Mor80].

The Harris detector is a *corner detector*: A corner is an area in the image where the visual signal changes abruptly in the two spatial directions, or, equivalently, where the gradient of the visual signal is simultaneously large in two orthogonal directions. This contrasts with *edges*: At an edge, the gradient is large in the direction that is perpendicular to the edge, but there is no gradient in the parallel direction[3]. We also mention the existence of a third main kind of interest point besides corners and edges: The *blobs* are found at the stable centers of circular, uniform regions (a region is uniform if its gradient is almost zero everywhere).

[3] In fact, the Harris detector is a combined corner and edge detector, so that it can also detect edges. This possibility is seldom considered in appearance-based vision because edges do not tend to be spatially localized, which may lead to unstable interest points. A noticeable exception is Jurie and Schmid's edge-based detector [JS04].

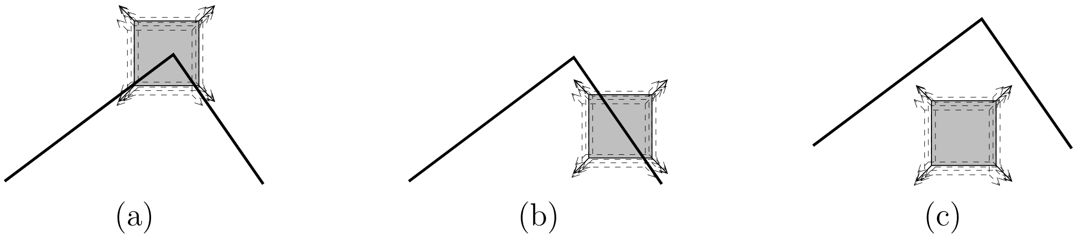


Figure 3.1: Basic idea of the Harris detector. (a) A corner is observed through a small window that is moved around. Shifting this window in *any direction* should give a significant change in the filter response. (b) When the window covers an edge, there is no change along the edge direction. (c) Over an uniform region, there is no change in any direction.

The idea of the Harris detector is to consider a window centered around a given pixel, then to shift the window in any direction and to measure the mean changes in the intensity that results from this shift (cf. Figure 3.1). If the mean change of the intensity is large in both directions, the detector assumes it has found a corner. This seminal idea was originally proposed by Moravec [Mor80].

Mathematically, given a shift (u, v), the mean change of intensity $C(x_0, y_0, u, v)$ that is induced by this shift when considering the pixel at location (x_0, y_0) is:

$$C(x_0, y_0, u, v) = \sum_{x,y} w(x - x_0, y - y_0) \left(I(x + u, y + v) - I(x, y)\right)^2 , \tag{3.1}$$

where $w(x, y)$ defines the considered window (representing the neighborhood). The Harris detector uses a two-dimensional Gaussian with standard deviation σ_I as the weighting window [HS88]:

$$w(x, y) = G(x, y; \sigma_I). \tag{3.2}$$

The value σ_I is known as the *integration scale* and determines the scale of the detected corners by dictating the size of the inspected window.

We now notice that the expression $I(x+u, y+v)$ in Equation 3.1 can be approximated using the gradient of the image I taken at point (x, y):

$$I(x + u, y + v) \approx I(x, y) + u\frac{\partial}{\partial x}I(x, y) + v\frac{\partial}{\partial y}I(x, y). \tag{3.3}$$

In practice, the image is smoothed by a two-dimensional Gaussian $G(x, y; \sigma_D)$ with standard deviation σ_D before the gradient is computed. This ensures an improved robustness to noise in the visual signal. The value σ_D is known as the *differentiation scale* and defines the smoothing extent. Its value is generally set proportional to the integration scale by the heuristic relation $\sigma_D = s\sigma_I$ with s comprised between 0.5 and 0.75 [MS02] (in practice, a good value for s is 0.7 [MS04]). It is common to use the shorthand notation $I_x(x, y)$ (resp. $I_y(x, y)$) to designate the value of the gradient of the smoothed image in the x-axis (resp. y-axis) direction[4]:

$$\begin{aligned} I_x &= \frac{\partial}{\partial x}(G(x, y; \sigma_D) \otimes I(x, y)), \text{ and} && (3.4) \\ I_y &= \frac{\partial}{\partial y}(G(x, y; \sigma_D) \otimes I(x, y)). && (3.5) \end{aligned}$$

Using matrix notation, Equation 3.1 can therefore be rewritten as:

$$\begin{aligned} C(x_0, y_0, u, v) &\approx \sum_{x,y} w(x - x_0, y - y_0) \left(uI_x(x, y) + vI_y(x, y)\right)^2 \\ &= \sum_{x,y} w(x - x_0, y - y_0) \\ &\qquad \left(u^2 I_x^2(x, y) + 2uvI_x(x, y)I_y(x, y) + v^2 I_y^2(x, y)\right) \\ &= \sum_{x,y} w(x - x_0, y - y_0) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} (x, y) \begin{bmatrix} u \\ v \end{bmatrix} \\ &= \begin{bmatrix} u & v \end{bmatrix} M(x_0, y_0) \begin{bmatrix} u \\ v \end{bmatrix}, && (3.6) \end{aligned}$$

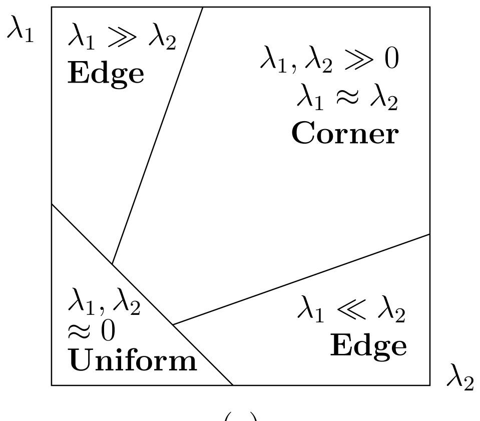
where $M(x_0, y_0)$ is the *auto-correlation matrix* at location (x_0, y_0). The matrix $M(x_0, y_0)$ is of size 2×2 for a fixed (x_0, y_0). It depends depends on the window function $w(x, y)$, and it can be evaluated through a two-dimensional convolution with a Gaussian kernel:

$$\begin{aligned} M(x_0, y_0) &= \sum_{x,y} w(x - x_0, y - y_0) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} (x, y) \\ &= \left(G(x, y; \sigma_I^2) \otimes \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} (x, y)\right) (x_0, y_0). && (3.7) \end{aligned}$$

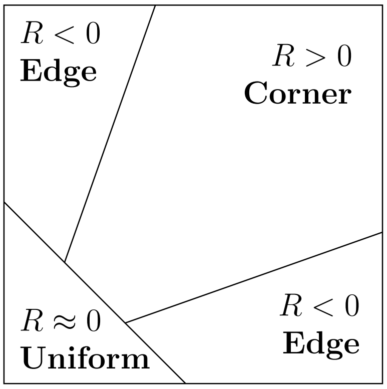
The auto-correlation matrix is independent of (u, v) and describes the gradient distribution in a local neighborhood of the point (x_0, y_0). It is also known as the *second moment matrix*.

Summarizing, Equation 3.6 is a bilinear approximation of the average change of intensity $C(x_0, y_0, u, v)$ at pixel (x_0, y_0) that is due to the shift (u, v). As a consequence, the eigenvectors of $M(x_0, y_0)$ represent the two principal curvatures of the gradient. The strength of the directions corresponding to the eigenvectors is directly related to their respective eigenvalues λ_1 and λ_2. Thus, when the eigenvalues λ_1, λ_2 are both large, then the intensity change $C(x_0, y_0, u, v)$ is large in all directions, which in turn implies with good confidence that the window is over a corner. On the other hand, if one eigenvalue dominates, the area is considered as an edge; and if both eigenvalues are negligible, the area is considered uniform. This is depicted in Figure 3.2 (a).

[4] In the following formulas, the operator $\otimes$ corresponds to the 2D discrete convolution.


λ_1
$\lambda_1 \gg \lambda_2$
Edge
$\lambda_1, \lambda_2 \gg 0$
$\lambda_1 \approx \lambda_2$
Corner
λ_1, λ_2
≈ 0
Uniform
$\lambda_1 \ll \lambda_2$
Edge
λ_2

(a)



(b)

Figure 3.2: Analysis of cornerness in the Harris detector. (a) Classification of an image point (x_0, y_0) using the eigenvalues of $M(x_0, y_0)$. (b) The decision boundaries based on the cornerness measure $R(x_0, y_0)$ that has been proposed by Harris and Stephens. Their slope is parametrized by the k constant.

Finally, Harris and Stephens provide an heuristic rule to decide when the eigenvalues are sufficiently large to induce a corner [HS88]. They consider the *cornerness measure* that is defined by the expression:

$$\begin{aligned} R(x_0, y_0) &= \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 && (3.8) \\ &= \det(M(x_0, y_0)) - k\,\mathrm{trace}(M(x_0, y_0))^2, && (3.9) \end{aligned}$$

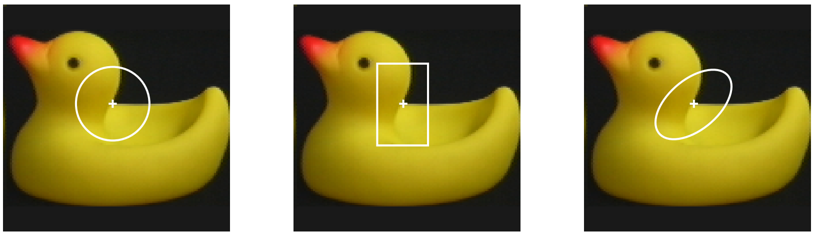
where k is empirically determined constant that lies in the interval $[0.04, 0.06]$ (Harris and Stephens suggest $k = 0.04$). When the two eigenvalues are large, the product dominates the sum and the cornerness is positive. When one eigenvalue is large with respect to the other, the sum dominates and the cornerness is negative. When the two eigenvalues are small, the cornerness is about zero. This leads to the decision boundaries that are represented in Figure 3.2 (b). Equation 3.9 can be used to compute the cornerness without achieving the costly computation of the eigenvalues.

Accordingly, the Harris detector computes the value of $R(x_0, y_0)$ at each pixel (x_0, y_0) of the image. If this value is above a fixed threshold, then the Harris detector considers (x_0, y_0) as the location of an interest point, and selects a neighborhood that is centered around (x_0, y_0) and whose size is related to the integration scale σ_I.[5] It can be shown that the Harris detector is invariant to rotation, and that it is robust to noise and illumination conditions [SMB00]. This quality is mainly due to the fact that the computation of the cornerness only implies evaluating first-order derivatives.

3.3.2 Interest Point Detectors

A possible mathematical definition of an interest point detector is:

[5] A non-maximum suppression on a 3×3 mask is also used to keep only local maxima [SMB00].



(a) (b) (c)

Figure 3.3: Commonly used support window spaces: (a) a circular neighborhood, (b) a rectangular neighborhood, and (c) an elliptic neighborhood.

Definition 3.3. An *interest point detector* $\mathcal{D}_L : S \mapsto \mathcal{P}(\mathbb{R}^2 \times W)$ is a mapping from the set of images S to the power set of interest points. An *interest point* is a triple $(x, y, w) \in \mathbb{R}^2 \times W$, where W is the *support window space*. It is required that $\mathcal{D}_L(s)$ is finite for each image $s \in S$.

An interest point detector converts an image $s \in S$ to a set of triples (x_i, y_i, w_i) describing the various interest points that have been detected in the image s. The location (x_i, y_i) of each interest point is accompanied with an information $w_i \in W$ that describes the shape of the neighborhood that has triggered the detection of the interest point (the component w_i is often referred to as the *support region*).

Remark 3.4. In general, the process of detecting interest points is stochastic (cf. Remark 3.2). A stochastic interest point detector is a probabilistic mapping $\mathcal{D}_L : S \mapsto \Pi(\mathcal{P}(\mathbb{R}^2 \times W))$. Unless explicitly mentioned, we only consider deterministic interest point detectors, that are by far most common. □

Support Window Spaces

The definition of an interest point detector resorts to a support window space. Here is a description of the most common support window spaces that are used in the local-appearance paradigm (cf. Figure 3.3):

Circular neighborhood: The support window is a circular neighborhood. The corresponding support window space collapses to a single real number that captures the scale of the circular neighborhood: $W = \mathbb{R}$. The Harris detector notably uses this support window space. In the latter case, W contains the integration scale σ_I.

Rectangular neighborhood: In this case, the support window is a rectangular neighborhood centered around the interest point location. The rectangle is possibly rotated by a given angle. The support window space contains this angle together with the width and the height of the rectangle: $W = \mathbb{R}^2 \times [0, \pi[$.

Elliptic neighborhood: An elliptic neighborhood is a generalization of a circular neighborhood. It is delimited by an ellipse defining a non-uniform scaling (i.e. that is possibly different in each direction). The corresponding support window space is $W = \mathbb{R}^2 \times [0, \pi[$: An element in this support window space is a triple (a, b, θ), where a defines the length of the semimajor axis, b the length of the semiminor axis, and θ a rotation angle for the ellipse. Evidently, other parametrization of an ellipse may be used.

An example of result of an interest point detector on a real-world image is depicted in Figure 3.4.

Description of Widespread Detectors

There exists an abundant literature about interest point detectors. Schmid et al. provide an historical treatment [SMB00]. The most important detectors are described below:

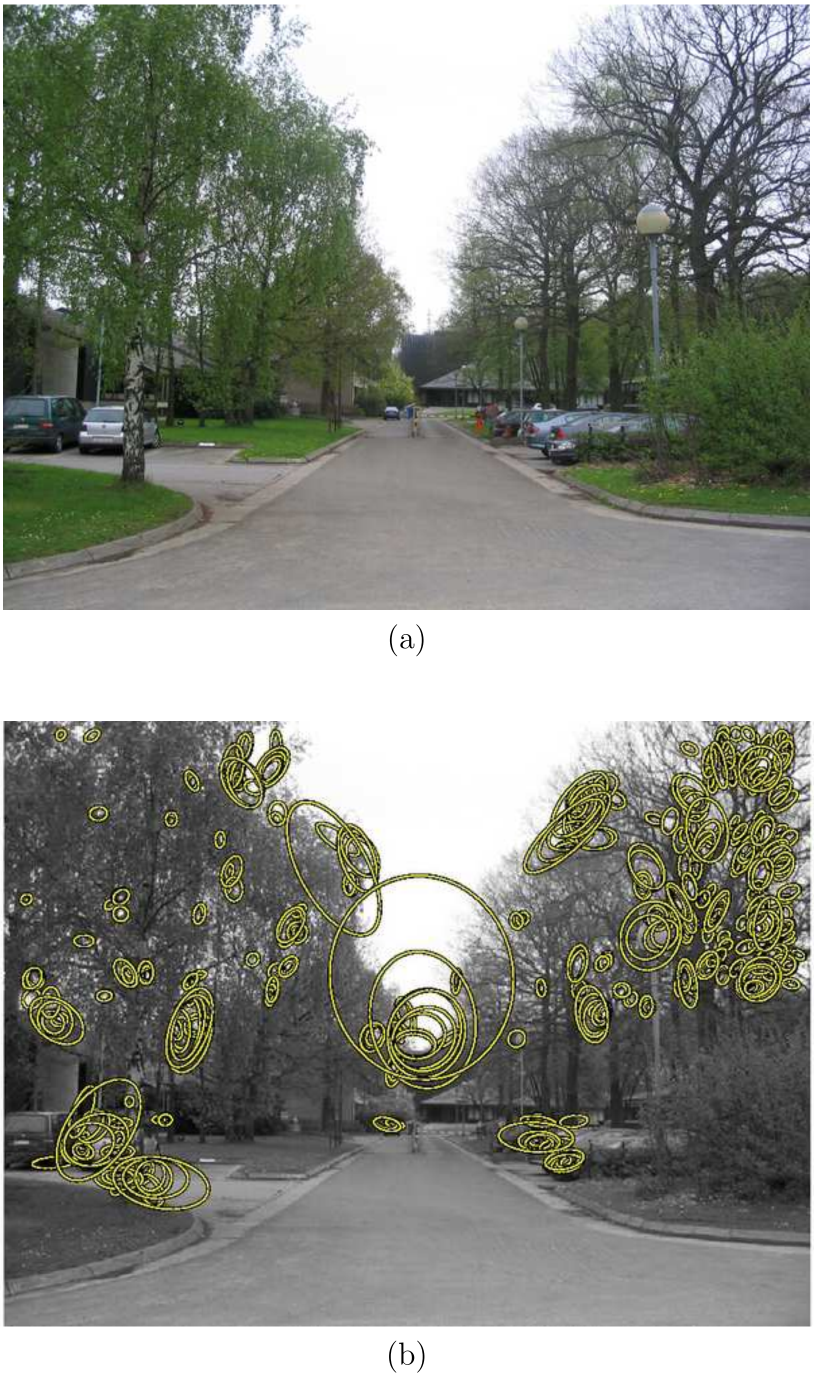
Fixed Grid: This is most basic interest point detector. It simply returns a set of interest points, each of them being sampled on a fixed, uniform grid. If the height of the image $s \in S$ is N pixels, if its width is M pixels, and if the grid has $m \times n$ cells, the interest point detector generates a set of $m \times n$ of interest points as follows:

$$\begin{aligned} \mathcal{D}_L(s) = \{(x_{ij}, y_{ij}, w_{ij}) \quad | \quad & i \in \{0, \dots, m-1\} \text{ and } j \in \{0, \dots, n-1\} \text{ and} \\ & (x_{ij}, y_{ij}) = \left(\frac{M(2i+1)}{2m}, \frac{N(2j+1)}{2n}\right)\Bigg\}. \end{aligned} \tag{3.10}$$

The fixed grid detector generally considers a fixed-sized rectangular neighborhood w_{ij} that is identical for all the cells in the grid. This detector is in general too simple for complex visual recognition tasks, but it may be sufficient for applications such as scene classification [FP05]. Indeed, it has the inherent advantage that the images are by definition densely covered.

Randomized detector: In a similar vein, Marée et al. [MGPW05] have proposed to use a fully randomized selection of interest points. This is the only stochastic interest point detector we evoke in this dissertation. In its basic version, the randomized detector selects k interest points uniformly at random, the neighborhood of which is a square w of fixed size. Using the definition of stochastic interest point detectors (cf. Remark 3.4), we can write:

$$\mathcal{D}_L(s, \{(x_1, y_1, w_1), \dots, (x_m, y_m, w_m)\}) =$$
$$\begin{cases} 0 & \text{if } m \neq k, \\ 0 & \text{if there exists } i \in \{1, \dots, m\} \text{ such that} \\ & x_i \notin \{0, \dots, M-1\} \text{ or } y_i \notin \{0, \dots, N-1\} \text{ or } w_i \neq w, \\ \left(\frac{1}{MN}\right)^k & \text{otherwise,} \end{cases}$$



(a)

(b)

Figure 3.4: Example of interest point detection. (a) An illustrative image that is taken from the database that has been collected for this research work [Jod05]. This database will be described in Section 5.1.5. (b) The result of the Harris-Affine interest point detector [MS04] on this image. Note that, in this case, the support window space consists of elliptic neighborhoods.

where $M \times N$ is the size of the image. More sophisticated versions of randomized detector select, for each interest point, a rectangular neighborhood with random size or/and random orientation. Marée et al. have showed remarkable performance of this approach for object classification [MGPW05], if it is combined with a powerful machine learning algorithm that is called Extra-Trees [GEW06] (cf. also Chapter 6).

Color Harris: As mentioned earlier, the Harris detector is at the core of many other interest point detectors. A first natural extension is to take the color of the images into account. This idea leads to the so-called *Color Harris detector* [GB01]. It is in essence identical to the Harris detector, but the auto-correlation matrix of Equation 3.7 is replaced as follows:

$$M = G(x, y; \sigma_I^2) \otimes \begin{bmatrix} R_x^2 + G_x^2 + B_x^2 & R_x R_y + G_x G_y + B_x B_y \\ R_x R_y + G_x G_y + B_x B_y & R_y^2 + G_y^2 + B_y^2 \end{bmatrix} (x, y), \tag{3.11}$$

where R_x (resp. R_y) denotes the first-order derivative over the x axis (resp. y axis) of the red color channel that has been previously smoothed by a Gaussian kernel of standard deviation σ_D (G_x, G_y, B_x and B_y are defined identically for the green and blue color channels). This way, the cornerness is computed across the color channels. According to Gouet and Boujemaa's comparison [GB02], this detector was the most stable interest point detector that makes use of color, with respect to rotational, illumination and viewpoint changes, as well as to noise. The Color Harris detector has been used for the tracking of soccer players [GHPV05].

Harris-Laplace: The Harris detector considers a single integration scale, and therefore fails in the presence of scale changes between images: It has been shown empirically that it can only cope with scale factors up to roughly 1.4 [SMB00]. This is problematic as soon as moving objects are filmed or the zoom level is modified. More fundamentally, a real-world object is typically composed of structures with various sizes. This is illustrated in Figure 3.5. Thus, the Harris detector is inherently unable to capture all the relevant structures of a scene.

This motivates the introduction of interest point detectors that can automatically select, for each location in the image, the scale that best suits the surrounding visual structure: Such a scale is called the *intrinsic scale* of the location. Lindeberg has extensively studied methods for automatic scale selection [Lin98]. When applied to scale selection of blobs, Lindeberg's approach consists in computing, for a finite set of scales $\{\sigma_1, \ldots, \sigma_k\}$, the response of the normalized Laplacian-of-Gaussian linear filter (which corresponds to the trace of the Hessian, normalized by the square of the scale):

$$\begin{aligned} r(x_0, y_0; \sigma_i) &= \sigma_i^2 \left| \frac{\partial^2}{\partial x \partial x} (G(\sigma_i) \otimes I)(x_0, y_0) + \frac{\partial^2}{\partial y \partial y} (G(\sigma_i) \otimes I)(x_0, y_0) \right| \\ &= \sigma_i^2 \left| I_{xx}(x_0, y_0) + I_{yy}(x_0, y_0) \right|, \end{aligned} \tag{3.12}$$

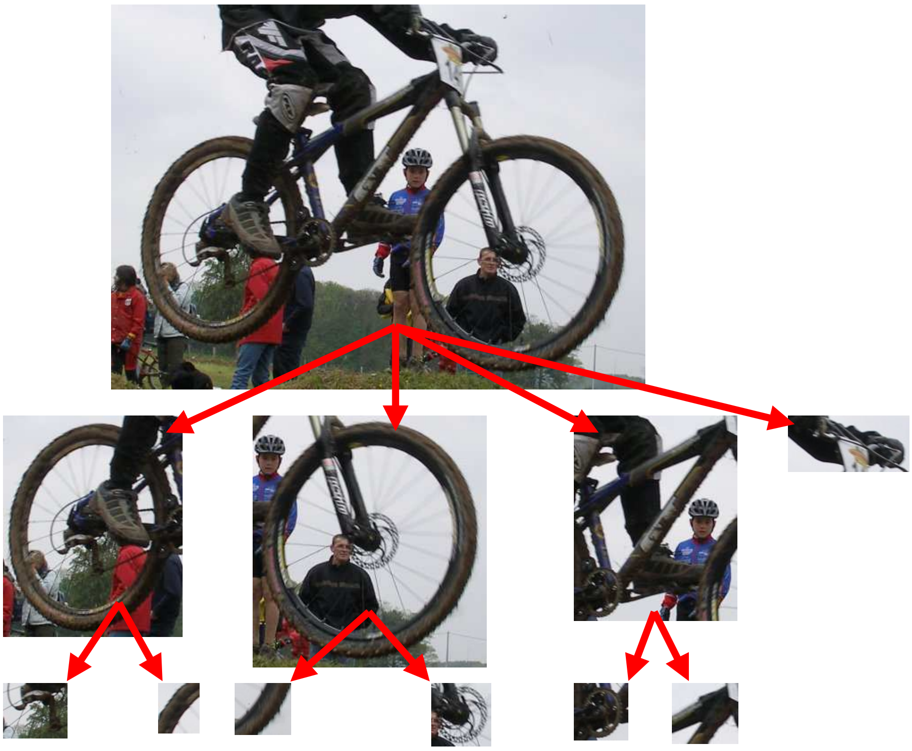
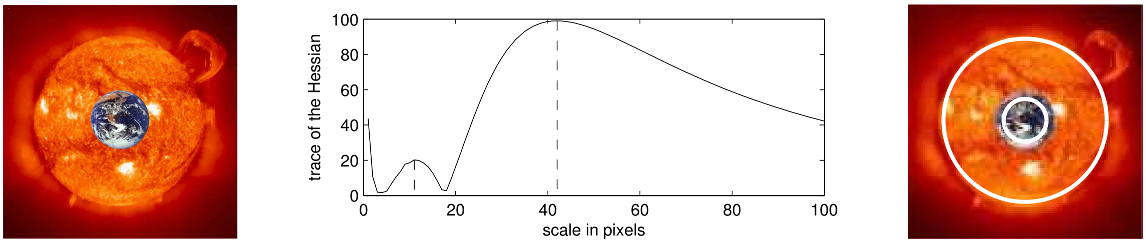


Figure 3.5: A bicycle can be hierarchically decomposed as a set of objects of varying size. Each of these constituting subparts should be observed at a scale that corresponds to its size, that is known as the *intrinsic scale* of the visual structure. (Reproduced with permission from Scalzo and Piater [SP06].)



(a) (b) (c)

Figure 3.6: Illustration of the computation of the intrinsic scale for blob features. (a) The source image the size of which is 121×121 pixels. (b) The scale-space signature $r(60, 60; \sigma)$ for the location at the center of the image. Each local maximum in this signature defines an intrinsic scale for this location: In this example, two intrinsic scales are detected at $\sigma = 11$ pixels and $\sigma = 42$ pixels. Thus, there are typically multiple intrinsic scales associated with each image location. (c) Two circles whose radii correspond to one computed intrinsic scale are overlaid on the source image. The smaller intrinsic scale is triggered by the blue blob corresponding to the earth, whereas the larger is induced by the orange blob of the sun. (The source image is reproduced with permission from Scalzo [Sca04].)

using notation similar to Equations 3.4 and 3.5. When this so-called *scale-space signature* attains a maximum over scale, the location is surrounded by a blob-like structure, the size of which corresponds to the size of the kernel. The process is illustrated in Figure 3.6. Although the isotropic kernel of Equation 3.12 is especially well adapted to blob structures, it also provides a good estimation of the intrinsic scale of corner and edge structures [MS04]. Interestingly enough, Marr has emphasized the importance of Laplacian-of-Gaussian in biological perception [Mar82b].

Automatic scale selection opens the path to *scale-invariant* interest point detectors. The rough idea of the *Harris-Laplace detector* is to use the intrinsic scales as the integration scale in the Harris detector [MS01a]. Sketchily, the Harris-Laplace detector involves two successive steps: (1) The Harris detector is applied at several integration scales[6], hence returning a set of interest points with circular neighborhood of varying size, then (2) the interest points that do not attain a maximum in the Laplacian-of-Gaussian scale-space signature are discarded.

Difference-of-Gaussians (DoG): Focusing on an efficient implementation, Lowe has developed the scale-invariant *DoG* detector, that is similar in spirit to Harris-Laplace but that approximates the Laplacian-of-Gaussian (LoG) of Equation 3.12 by a Difference-of-Gaussians (hence the name DoG) [Low99].

DoG is a close approximation of LoG, but it has the advantage that it can be evaluated at various scales efficiently. To this end, a pyramidal representation of the image is built. Each level in this pyramid is the smoothed version of the lower level, starting with the bottom level containing the source image. The top level corresponds to the coarsest scale, and is limited by the size of the image. This representation is known as the *Gaussian pyramid*, and can be evaluated efficiently. Then, the detector builds the so-called *DoG pyramid*: Each level of the DoG pyramid is obtained by subtracting two successive levels of the Gaussian pyramid [FP03, Section 9.2]. As the Laplacian is roughly equal to the difference of two Gaussians, each level of the DoG pyramid is an approximation of the LoG at the corresponding scale.

Lowe proposes to consider the DoG pyramid as a tensor with three dimensions that is parametrized by a triple (x, y, σ), and then to identify local 3D extrema in this tensor. Once a maximum is reached at coordinates (x_0, y_0, σ_0), the detector assumes it has found an interest point at location (x_0, y_0) with a circular neighborhood of size σ_0. The drawback of this approach is that the local maxima tend to also lay over edges. Thus, a cleanup step is necessary to discard these less stable interest points. On the other hand, using DoG instead of LoG considerably reduces the computation expense, making DoG suitable for real-time applications, at the price of a slight loss in accuracy.

[6] The considered integration scales are $1.4^i\sigma_0$, where i is an integer and σ_0 an initial scale.

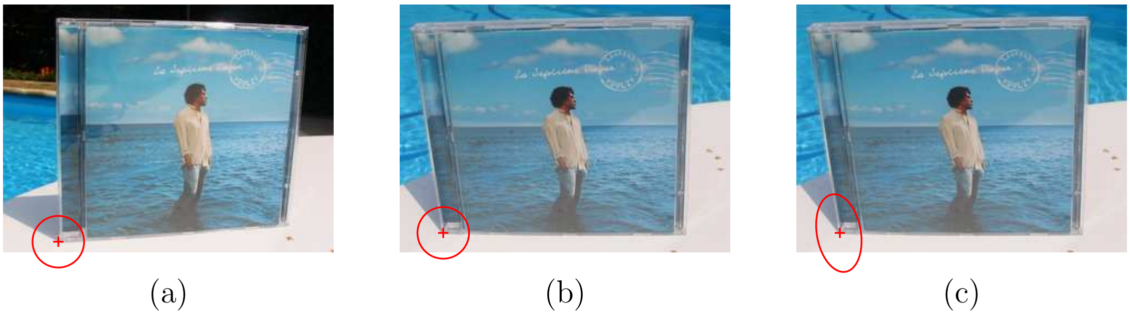


Figure 3.7: Two views of the same object under viewpoint changes (a,b). No circular neighborhood in view (b) can capture the same physical surface as the circular neighborhood in view (a) (notably the shadow), whereas the elliptic neighborhood in view (c) does. (This illustration is strongly inspired by Mikolajczyk et al. [MTS$^+$05].)

Harris-Affine: The Harris-Laplace and DoG detectors offer invariance to similarity transforms (translation, rotation and scale), but not to the changes in viewpoint. Indeed, a circular neighborhood cannot cope with the geometric deformations caused by a viewpoint change (cf. Figure 3.7). Nevertheless, if the object surface is almost planar and if the perspective effects are neglected, an anisotropic rescaling (i.e. an affinity) can approximate the projective transformation that results from the visual sensing process. As a consequence, recent work in local-appearance vision considers elliptic neighborhoods.

Harris-Affine is such an affine-invariant detector [MS04]. It first applies the Harris-Laplace detector so as to extract an initial set of interest points. Then, for each interest point, the following process is repeated: (1) The eigenvalues of the auto-correlation matrix are used to estimate the anisotropic shape of the elliptic neighborhood around the interest point (this is achieved through an iterative process that is due to Lindeberg and Baumberg [Lin98, Bau00]), and (2) the elliptic neighborhood is normalized to a circular neighborhood. The process stops when the eigenvalues of two successive auto-correlation matrices are equal, which means that the computed elliptic neighborhood is stable. If a fixed point cannot be reached, the interest point is discarded.

We have reviewed several interest point detectors that are based on the ideas of the Harris detector, and that have attracted a lot of attention over the last decade. Another useful detector is *Edge-Based Regions* (EBR) [TVG04] that is affine-invariant.

However, appearance-based vision might well be currently at a turning point. Indeed, recent work suggests that blob detectors are more stable and repeatable than cornerness-based detectors [MTS$^+$05]. Interest point detectors that select blob structures notably comprise the *Hessian detector* [Bea78], the *Hessian-Laplace detector* and the *Hessian-Affine detector* [GL96, Lin98, MS02]. These methods are the counterparts of the Harris-family detectors where the cornerness measure is replaced in favor of the determinant of the Hessian. The Hessian of the image is built with second-order derivatives of the image, beforehand smoothed by a Gaussian. The Hessian fires on blobs because of its isotropic "Mexican hat" shape.

A very interesting evolution of the Hessian-Laplace detector is the *Fast-Hessian detector* [BTVG06b]. Following the same line of ideas as the DoG detector, Fast-Hessian approximates the Hessian-of-Gaussians filter by box filters. This allows a much faster computation if integral images [VJ01] are internally used[7], without losing much accuracy. Other examples of detectors for blob-like structures are *Intensity Extrema-Based Regions* (IBR) [TVG04], *Maximally Stable Extremal Regions* (MSER) [MCUP04], and *Salient Regions* [Gil98, KZB04]. Of the above, IBR and MSER are fully affine-invariant, whereas Fast-Hessian and Salient Regions are only scale-invariant.

Comparison of the Detectors

Following the criteria about good visual features that were presented in Section 3.1.1, we deduce that the ideal interest point detector has the following properties:

- *robustness* (cf. Section 3.1.1),
- *repeatability* (two distinct views of the same scene should share a high percentage of common interest points),
- *quantity* (the objects of interest should be densely covered, even if their size in the image is small),
- *accuracy* (the location of an interest point should be accurate), and
- *simplicity* (cf. Section 3.1.1).

Again, the choice of an interest point detector must be balanced by the computer vision task to be solved. For example, the interest points that are used in camera calibration, stereovision or structure from motion must be accurate (if possible, with sub-pixel precision). On the other hand, quantity is much more important when dealing with object recognition.

As a consequence, performance evaluation of detectors has gained more and more importance in modern computer vision. To make things simple, Mikolajczyk et al. suggest that the highest repeatability among the affine-invariant detectors is obtained by MSER, closely followed by Hessian-Affine [MTS$^+$05]. Unfortunately, ensuring the affine invariance demands much more complex algorithms and involves a high computational burden, which might not systematically be compensated by the gain in repeatability. In practice, the scale-invariant DoG is probably the most versatile detector, and is pretty fast with respect to Harris-Laplace and Harris-Affine [MS04]. The recent Fast-Hessian detector might prove to be a powerful, even faster alternative to DoG [BTVG06b]. Finally, as long as scale invariance and affine invariance are not required, the Harris detector is certainly the best choice [SMB00].

[7] Typical speedup is of order 4. An image of size 800 × 640 can be treated in 120ms.

3.3.3 Local Descriptors

As explained in Section 3.3, once interest points have been selected, a local descriptor generator is used to convert the neighborhood of this point as a vector of real-valued attributes:

Definition 3.5. A *local descriptor generator* $\mathcal{G}_L : S \times (\mathbb{R}^2 \times W) \mapsto V$ is a deterministic mapping from an interest point to a vector of real numbers.

Not all local descriptor generators can use the full expressiveness of the support windows that are spanned by the interest point detector. However, it is always possible to convert a support window space to another one at the price of some approximation (e.g. an elliptic neighborhood can be normalized to a circular one, a rectangular neighborhood can be approximated by an elliptic one, and so on).

Description of Widespread Descriptors

As for interest point detectors, a large number of local descriptor generators have been proposed over years in the literature. We now review the most useful:

Adapting global-appearance methods: Any global-appearance method can be directly adapted so as to describe the content of the neighborhood around an interest point (cf. Section 3.2).

The simplest description technique is therefore to normalize the neighborhood of the interest point to a square of fixed size, and to directly use the raw pixel values as a local descriptor. Following this idea, Marée et al. have applied *random subwindows* [MGPW05] to image classification. A random subwindow is a raw image patch the size of which is typically 11 × 11 pixels, and that is selected through the randomized interest point detector. In their approach, the color information of the patches is not discarded, and the image patches are generally converted to the HSV colorspace. This idea was motivated by earlier work showing that patches of size 5 × 5 can lead to performance that are close to other state-of-the-art local description techniques [EC04]. Scalzo and Piater also use raw image patches for the probabilistic representation of a hierarchy of visual features [SP06].

Similarly, several authors have considered the application of PCA for describing the local appearance (cf. Section 3.2.2). The earliest work on this topic defines *eigenwindows* [OI97], that carry out a Principal Component Analysis on graylevel patches of size 15×15. This topic was further investigated by Colin de Verdière and Crowley [CdVC98], as well as by Paredes et al. [PPJV01].

Finally, color histograms have also be used to describe the neighborhood of an interest point [EM95].

Differential descriptors: Another important family of local descriptors is *differential descriptors*. They can be applied to circular neighborhoods or to elliptic neighborhoods, and are based on the evaluation of the derivatives of the image.

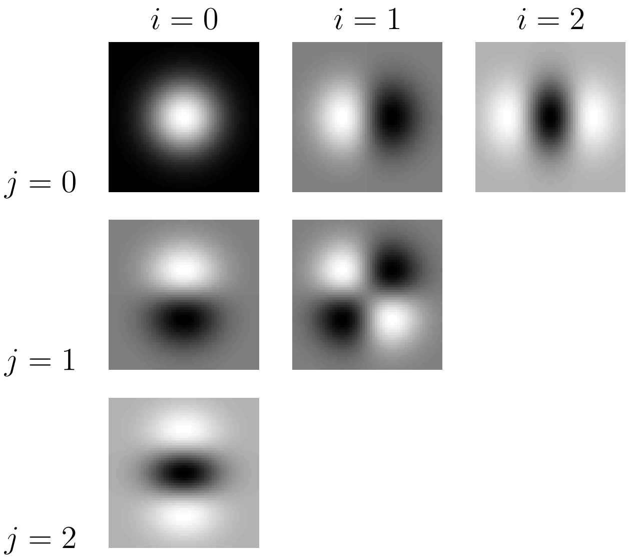


Figure 3.8: The Gaussian derivatives $\frac{\delta^{i+j}}{\delta x^i \delta y^j} G$ up to second order. In practice, derivatives of order higher than 3 are mostly useless, due to their high sensitivity to noise.

Indeed, following the Taylor expansion applied to the image intensity surface, the neighborhood of an interest point can be represented as the derivatives up to a given order. If an elliptic neighborhood is to be characterized, it is first normalized to a circular neighborhood.

Following this reasoning, the so-called *local jet* descriptor is a truncated vector that comprises all the differential responses up to a given order [KvD87]. In practice, to get a stable local jet, the image is smoothed by a Gaussian filter before computing the derivatives themselves. By the linearity of 2D convolution, this process is equivalent to filtering the image with a Gaussian derivative (cf. Figure 3.8). The size of the Gaussian kernel is chosen so as to correspond to the radius of this neighborhood. Using the notation of Equations 3.4, 3.5 and 3.12, the local jet up to second order is:

$$\begin{pmatrix} I \\ I_x \\ I_y \\ I_{xx} \\ I_{xy} \\ I_{yy} \end{pmatrix} . \tag{3.13}$$

As such, the local jet is not invariant to rotation. This invariance can be ensured by two different approaches: (1) The Gaussian derivatives are rotated (*steered*) to aim in the direction of the gradient before being applied, leading to *steerable filters* [FA91][8]; or (2) the components of the local jet are

[8] Steerable filters might turn out to be problematic, if the gradient is ill-conditioned. This is notably the case on uniform or symmetrical circular neighborhoods.

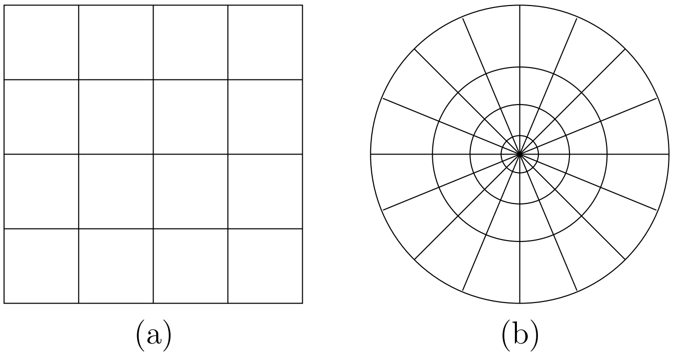


Figure 3.9: Decomposition of the neighborhood used (a) by the SIFT descriptor, and (b) by the Shape Context descriptor.

combined so as to obtain rotation invariance, giving rise to *differential invariants* [FtHRKV91]. The differential invariants up to second order are given by:

$$\left(\begin{array}{cl} I & \leftrightarrow \textbf{intensity} \\ I_x^2 + I_y^2 & \leftrightarrow \textbf{gradient magnitude} \\ I_{xx} + I_{yy} & \leftrightarrow \textbf{Laplacian} \\ I_x^2 I_{xx} + 2 I_{xy} I_x I_y + I_y^2 I_{yy} & \\ I_{xx}^2 + 2 I_{xy}^2 + I_{yy}^2 & \end{array} \right) . \qquad (3.14)$$

Differential descriptors have also been generalized to color images, leading to *color differential invariants* that are 8-dimensional [MGD98, GB02]. Thanks to a careful exploitation of the color information, it is possible though not mandatory to stop the expansion of color differential invariant at the first order, which is beneficial with respect to noise. Color Differential Invariants have been successfully applied to soccer player tracking in conjunction with the Color Harris detector [GHPV05]: As soccer teams can only be distinguished by their colors, it is indeed required to take color into account in this context.

Filter banks: The neighborhood of an interest point can also be described by its texture, thanks to the outputs of a set of linear filters [FP03, Section 9.1]. This approach is evidently closely related to texture histograms (cf. Section 3.2.3). Commonly used sets of filters include Gabor filters [MM96], wavelets [POP98], as well as custom-engineered filter banks that are tuned for specific applications [RH99]. Despite their historical popularity, the filter banks do not always provide the best method for image description. For instance, raw pixel values can be more effective than filter outputs for texture classification [VZ03].

SIFT (Scale-Invariant Feature Transform): So far, the spatial distribution of the visual attributes (such as raw pixels, PCA or filter outputs) around the interest point have not been fully taken into account. In recent literature, very promising results have been achieved by so-called *distribution-based descriptors* that subdivide the considered neighborhood and compute visual attributes separately inside each subregion [MS05]. In this way, a compromise is achieved between geometric invariance on the one hand and greater discriminative power on the other hand.

Currently, the *Scale-Invariant Feature Transform* (SIFT) is certainly the most popular local descriptor [Low99, Low04]. SIFT is precisely a distribution-based descriptor: It divides a square neighborhood into a 4×4 grid and computes a histogram of local gradient orientations in each subregion. This is illustrated in Figure 3.9 (a). Gradient orientations are discretized in eight bins[9], which results in a vector of 128 real numbers. Using histograms provides stability against deformations of the image pattern, whereas the subdivision prevents the potential loss of spatial information.

Gradient Localization and Orientation Histograms (GLOH) [MS05] and *PCA-SIFT* [KS04a] are two evolutions of SIFT. The former is 128-dimensional and applies PCA to a log-polar location grid, whereas the latter is 36-dimensional and applies PCA to the normalized gradient image.

Finally, the authors of the Fast-Hessian detector have recently introduced the *Speeded Up Robust Feature* (SURF) descriptor [BTVG06b]. Just like the Fast-Hessian detector is an approximate version of the DoG detector, the SURF descriptor is an approximation of the SIFT descriptor. The computation can be carried out about 3-4 times faster (the running time on a 800×640 image is 350ms), and might as such be used in real-time applications.

Shape Context: Another important distribution-based descriptor is *Shape Context* [BMP02] that is based on the Canny edge detector [Can86]. Shape Context uses a log-polar decomposition of a circular image region, and counts edge points in each spatial bin (cf. Figure 3.9 (b)). The corresponding descriptor is 36-dimensional.

The aforementioned descriptors are probably the most widespread. Mikolajczyk and Schmid [MS05] succinctly describe other descriptors: *Gradient Moment Invariants*, *Spin Image*, *Complex Filters* and *Cross Correlation*. For the sake of completeness, we also mention the existence of *Color Moment Invariants* [SSTVG03], *Local Affine Frames* [OM02, OM03] and *Order Type* [TC04].

Comparison of the Descriptors

The ideal interest point detector has the following properties (these properties have been already discussed in Section 3.1.1): *distinctiveness*, *robustness*, *simplicity* and *compactness*. Mikolajczyk and Schmid provide an in-depth comparison of the performance of various descriptors [MS03, MS05]. The best overall performance is obtained by SIFT and GLOH, so that the current *de facto* standard for local-appearance vision is to use the SIFT descriptor. Unfortunately, SIFT and GLOH are high-dimensional (which makes similarity calculations costly) and require sophisticated, computationally expensive algorithms. This makes them inapplicable to robotic and/or real-time

[9] A linear interpolation is used at this step to obtain smoothed weighted histograms.

applications[10]. Among the low-dimensional descriptors, steerable filters and gradient moment invariants are the best choices.

It is important to note that Mikolajczyk and Schmid's performance evaluation does not include descriptors based on the raw pixels of the images [MGPW05]. Furthermore, the promising SURF was not yet developed at the time this comparison was written [BTVG06b]. Preliminary experimental results show that SURF outperforms SIFT in repeatability and recall, while being easier to implement. Thus, future research in local-appearance vision might favor these two recent algorithms.

3.3.4 Local-Appearance Feature Generators

If a local descriptor generator is used in conjunction with an interest point detector, a visual feature generator is obtained (cf. Definition 3.1). As such a generator is founded on the local-appearance paradigm, it is known as a local-appearance feature generator:

Definition 3.6. Let:

- $\mathcal{D}_L : S \mapsto \mathcal{P}(\mathbb{R}^2 \times W)$ be an interest point detector (cf. Definition 3.3), and
- $\mathcal{G}_L : S \times (\mathbb{R}^2 \times W) \mapsto V$ be a local descriptor generator (cf. Definition 3.5).

Then, the *local-appearance feature generator* $\mathcal{G}_V : S \mapsto \mathcal{P}(V)$ that is induced by $\mathcal{D}_L$ and $\mathcal{G}_L$ is defined as:

$$\mathcal{G}_V(s) = \bigcup_{(x,y,w)\in\mathcal{D}_L(s)} \mathcal{G}_L(s,(x,y,w)). \tag{3.15}$$

Remark 3.7. Publicly available implementations of interest point detectors and local descriptors are available [Low05, Dor05, Mik06, BTVG06a]. □

3.4 Exploiting Visual Features

Once a finite set of visual features has been extracted through a visual feature generator, the third step consists in exploiting this information for a given computer vision task. This can be achieved through at least two distinct approaches: by *matching the visual features*, or by *learning a predictive model*.

Let us consider the epitome of image classification. This computer vision task consists in attributing a class label $c(s) \in C$ to a given image $s \in S$. The finite set C contains the possible labels. Although more sophisticated approaches exist, one simple method to solve this task consists in two steps:

[10] This claim must be nuanced, as SIFT extraction has been implemented on a Field-Programmable Gate Array (FPGA) [SNJM04].

Learning phase: Assume we are given a database of learning pairs $\langle s_i, c_i\rangle \in S \times C$ that maps a set of images to their respective ground-truth class labels. The learning phase applies the following steps: (1) For each image s_i in the database, a set

$$\left\{\boldsymbol{v}_1^{(i)}, \ldots, \boldsymbol{v}_{k_i}^{(i)}\right\} \tag{3.16}$$

of k_i visual features is extracted through the visual feature generator $\mathcal{G}_V$; and (2) a *visual feature database* is built, that maps each extracted visual feature to the class label assigned to the image that originally contained this feature:

$$\left\{\langle \boldsymbol{v}, c\rangle \mid (\exists i)\ (\exists j \in k_i)\ \boldsymbol{v} = \boldsymbol{v}_j^{(i)} \text{ and } c = c_i\right\}. \tag{3.17}$$

Classification phase: When classifying an unseen image $s \in S$, the following algorithm is carried out: (1) A set of visual features $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ is generated from s; (2) each of these visual features is assigned a class label by querying the visual feature database, so that each visual feature votes for one class label; and (3) the unseen image is assigned with the class label that has obtained most votes.

The keystone of the algorithm above is the querying of the visual feature database at the second step of the classification phase. This step assigns one class label to a single visual feature. Such a step is not unique to image classification: It is in general present in all the vision systems that make use of local appearance. The way this query is achieved allows to distinguish between two different, general approaches to the exploitation of visual features, which have already been mentioned at the beginning of this section:

Matching the visual features: The first way to query the visual feature database consists in scanning the raw database $\{\langle \boldsymbol{v}_l, c_l\rangle\}$ so as to find the feature $\boldsymbol{v}_{l^*} \in V$ that best matches an input visual feature $\boldsymbol{v}$. Then, the label c_{l^*} of this best feature $\boldsymbol{v}_{l^*}$ is assigned to the input feature. To this end, a metric $d(\boldsymbol{x}, \boldsymbol{y})$ must have been defined on the visual feature space V, and the best visual feature is defined as the one that minimizes the distance with the input feature:

$$c(\boldsymbol{v}) = c_{l^*}, \text{ with } l^* = \operatorname*{argmin}_l d(\boldsymbol{v}_l, \boldsymbol{v}). \tag{3.18}$$

Euclidean distance for $d(\boldsymbol{x}, \boldsymbol{y})$ can be used in this context [OM02, KS04a]. However, this distance may be insufficient, as it cannot cope with scale differences between the variables, neither with the interdependency that possibly links several variables. Therefore, the Mahalanobis distance [SM97, SSTVG03] and the tangent distance [SLDV98, KMND04] are often considered.

As far as histogram-based visual features are used (cf. Section 3.2.3), specific similarity measures have been designed. These measures notably include *Histogram Intersection* [SB91], *Sum of Squared Distance* [CM95], χ^2-*statistic* [SC96] and *Earth Mover's Distance* [RTG00]. Rubner et al. compare these similarity measures for histograms [RPTB01].

From a general perspective, *matching the features* conceptually corresponds to a *nearest neighbor* retrieval. Evidently, variants of nearest neighbor can be used, such as k-nearest neighbor retrieval: This retrieval method outputs the class that is the most represented among the k visual features that are the nearest from the input visual feature.

We finally note that efficient data structures for nearest neighbor retrieval can be used for achieving the minimization of Equation 3.18. For instance, Schmid and Mohr [SM97] use *k-d trees* [Sam84], Georgescu et al. [GSM03] resort to *Locality Sensitive Hashing* [IM98, LMGY04], whereas Obdržálek and Matas propose a decision-tree based indexing [OM05].

Learning a predictive model: The second possibility to query the visual feature database consists in generating a predictive model from the database through a suitable machine learning algorithm. The computed model is thereafter used to classify unseen visual features. The introduction of a metric is not required here, as the learning and the prediction phases do not resort to the raw visual feature database.

For example, Marée et al. [MGPW05] build a classification model for raw image patches using the Extra-Trees learning algorithm [GEW06]. As already mentioned, this approach to image classification reaches performance that are closed to state-of-the-art methods based on the matching of visual features. A closely related paper uses *Randomized Trees* [LF06].

Earlier work on model learning often focuses on *Support Vector Machines* and *Neural Networks*. Marée reviews these methods in the case raw image patches are used as global-appearance visual features [Mar05, Section 3.2.3]. Support Vector Machines have also been used to classify local-appearance visual features [CWN04, EC04, ZMLS06].

3.5 Summary

In this chapter, techniques for creating a mid-level representation of images were discussed. Emphasis was put on appearance-based visual features, and, more specifically, on local-appearance features. We looked at the widespread Harris detector in detail. More sophisticated interest point detectors were then discussed. Finally, we presented several local description algorithms and the way visual features are exploited in appearance-based vision.

Using a local-appearance approach for closed-loop learning of visual tasks in uncommitted environment seems to be a promising approach. Indeed, the learning agent has to focus its attention on discriminative visual cues, but without being aware of what cue is background clutter, and without knowing which part of the observed objects are occluded. As a consequence, we choose to develop RL algorithms that take advantage of local-appearance visual features.

CHAPTER

FOUR

Reinforcement Learning of Visual Classes

Chapters 2 and 3 have respectively proposed an introduction to reinforcement learning and to appearance-based vision. We now combine the ideas of these two chapters for closed-loop learning of visual tasks. Similarly to local-appearance vision (cf. Section 3.4), there exist at least two possible approaches for exploiting visual features in the context of reinforcement learning:

1. *Once a metric and a threshold on this metric are defined in the visual feature space V, any visual feature splits the visual space S in two parts: Those images that exhibit this feature, and the others. As a consequence, visual features can be used to add structure to the visual space. By testing the presence of selected, highly informative visual features in the input images, a bunch of* visual classes *can be built. Thus, the complex, high-dimensional visual space is mapped to a finite set of visual classes that can hopefully be much smaller than the original set of possible images. Thereby, standard RL algorithms can be used to extract an optimal control policy. Such a task-driven discretization of the visual space subscribes to a* top-down philosophy*, and is adopted by the* Reinforcement Learning of Visual Classes *(RLVC) algorithm that is discussed in the current chapter.*

2. *Another possibility consists in approximating the optimal state-action value function $Q^*(s, a)$ by a regression model that takes the raw visual features as input. Thus, there is no need to introduce a metric on the visual feature space V. In this case, the visual features are used as they are, without comparing them to a selected subset of highly informative visual features. This* bottom-up philosophy *will be exploited in* Visual Approximate Policy Iteration *(V-API) in Chapter 6.*

4.1 Features in the Perceptual Space

As stated in Chapter 2, the objective of reinforcement learning is to learn an optimal percept-to-action mapping $\pi^* : S \mapsto A$ (cf. Theorem 2.15 and Definition 2.16). Such a mapping directly links the state space S to the action space A. To emphasize the fact that the agent takes its decisions by resorting only to the current state S, the state space S will be equally referred to as the *perceptual space*. Similarly, any state $s \in S$ can be called a *percept*. Chapter 3 has described how it is possible to define visual features on a perceptual space that is constituted by images. We now generalize the notion of a visual feature to non-visual perceptual spaces. The algorithm RLVC will be defined using these more general notions, so that RLVC can be applied to vision-for-action tasks as well as to other kind of control problems.

4.1.1 Visual Features Exhibited by Images

As mentioned above, RLVC exploits the visual features in a way that is inspired by computer vision applications that match visual features (cf. Section 3.4). Thus, RLVC assumes that a metric $d(\boldsymbol{x}, \boldsymbol{y})$ has been defined on the visual feature space V. Furthermore, to be able to match two visual features, a fixed threshold ε on this metric is introduced:

Definition 4.1. Let $\boldsymbol{v_1}, \boldsymbol{v_2} \in V$ be two visual features. These visual features are *matched* with respect to to the metric $d(\boldsymbol{x}, \boldsymbol{y})$ and to the threshold $\varepsilon \in \mathbb{R}_0^+$ if:

$$d(\boldsymbol{v_1}, \boldsymbol{v_2}) < \varepsilon. \tag{4.1}$$

Thanks to the concept of matched visual features, an image is said to exhibit a visual feature if a matching visual feature can be found at one of its interest points by the visual feature generator $\mathcal{G}_V$ (cf. Definition 3.1):

Definition 4.2. Let $s \in S$ be an image, and let $\boldsymbol{v} \in V$ be a visual feature. The image s is said to *exhibit* the visual feature $\boldsymbol{v}$ if:

$$(\exists \boldsymbol{v'} \in \mathcal{G}_V(s))\ d(\boldsymbol{v}, \boldsymbol{v'}) < \varepsilon. \tag{4.2}$$

This definition naturally extends to a *visual feature detector* that tests whether a given image exhibits a given visual feature or not:

Definition 4.3. A *visual feature detector* is a mapping $\mathcal{D}_V : S \times V \mapsto \mathcal{B}$ that associates a Boolean number to a pair that is composed of a visual percept $s \in S$ and a visual feature $\boldsymbol{v} \in V$. The visual feature detector is defined by the following relation:

$$\mathcal{D}_V(s, \boldsymbol{v}) = \textbf{true} \text{ if and only if } s \text{ exhibits } \boldsymbol{v}. \tag{4.3}$$

Although these notions come from an analysis of appearance-based vision, they can be naturally extended to non-visual perceptual spaces.

4.1.2 General Perceptual Features

Any visual feature $\boldsymbol{v} \in V$ indeed acts as a *binary classifier* that maps any image to two classes: those images that exhibit $\boldsymbol{v}$ (cf. Definition 4.2), and the others. Of course, the concept of a feature is not limited to visual spaces: Visual features are nothing else but a special kind of the more abstract notion of *perceptual features*. A perceptual feature is anything that can be either present or absent from a percept. This leads us to the following definition:

Definition 4.4. A *perceptual feature detector* $\mathcal{D}_S : S \times F_S \mapsto \mathcal{B}$ is a mapping that associates a Boolean to any pair that consists of a percept and of a perceptual feature[1]. F_S is the *perceptual feature space*, that is possibly infinite.

Below follow some examples of useful perceptual feature detectors for some common perceptual spaces.

Visual Spaces

In this case, the perceptual feature space F_S directly corresponds to the visual feature space V. The corresponding perceptual feature detector tests if the given image exhibits the given visual feature at one of its interest points (cf. Definition 4.2):

$$\mathcal{D}_S(s, \boldsymbol{v}) = \textbf{true} \text{ if and only if } (\exists \boldsymbol{v'} \in \mathcal{G}_V(s))\ d(\boldsymbol{v}, \boldsymbol{v'}) < \varepsilon. \tag{4.4}$$

Continuous Spaces

Continuous perceptual spaces have received by far the most attention in the RL literature. In this case, S corresponds to $\mathbb{R}^n$ for some $n > 0$. Inspired by the theory of decision tree induction [BFS84], a perceptual feature in such continuous spaces is defined as a test that checks if a given component of the vector encoding the percept is smaller than a given threshold:

$$\begin{aligned} F_S &= \mathbb{R} \times \{1, \ldots, n\}, & (4.5) \\ \mathcal{D}_S(\boldsymbol{s}, (t, i)) &= \textbf{true} \text{ if and only if } s_i < t. & (4.6) \end{aligned}$$

The same kind of features has notably been used by Munos and Moore in *Variable Resolution Grids* [MM02].

Perceptual Spaces with Binary Numbers

Some work about reinforcement learning considers learning agents whose percepts are binary numbers. For example, Chapman and Kaelbling have designed an RL algorithm that can be fed with very simple, strongly structured, monochrome images from the video game Amazon [CK91]. An image is encoded as a fixed-length binary number that is formed by the concatenation of the value of its pixels. More recently, Porta and Celaya consider environments in which the perceptual space is a fixed-sized vector of variables with a finite variation domain [PC05]. Such percepts contain the Boolean answers of a bank of binary tests that check a given property of the

[1] The subscript S of the notation $\mathcal{D}_S$ and F_S emphasizes the fact that these features are defined on the perceptual space S. This convention will be useful in Chapter 7.

percept.[2] Thus, such perceptual spaces that are composed of binary numbers can be defined as:

$$S = \bigcup_{n \in \mathbb{N}} \mathcal{B}^n, \tag{4.7}$$

where $\mathcal{B}$ designates the set of Boolean numbers.

In this context, a suitable perceptual feature indicates the position of one bit in the binary number. The perceptual feature detector tests whether the indicated bit is set in the given percept. Formally, we obtain the following definitions:

$$\begin{aligned} F_S &= \mathbb{N}, & (4.8) \\ \mathcal{D}_S(b_{n-1} \dots b_0, i) &= \textbf{true} \text{ if and only if } (i < n \text{ and } b_i = \textbf{true}) \,. & (4.9) \end{aligned}$$

Note that if the indicated bit is absent from the binary number, it is assumed to be **false**. This is basically the same convention as in the standard positional number systems.

4.1.3 Perceptual Feature Generators

Besides the visual feature detector, the other important concept from appearance-based vision is the visual feature generator. Once again, it is possible to naturally extend this concept to non-visual perceptual spaces:

Definition 4.5. A *perceptual feature generator* $\mathcal{G}_S : \mathcal{P}(S) \mapsto \mathcal{P}(F_S)$ is a mapping that associates to each set of percepts, the set of all the perceptual features that are present in one of these percepts. For computational tractability, it is required that for each finite set $\{s_1, \dots, s_m\}$ of percepts, $\mathcal{G}_S(\{s_1, \dots, s_m\})$ is finite.

Note that it is allowed for a perceptual feature generator to generate an empty set of perceptual features. This can happen, for example, when an image is uniform, in which case no salient interest point can be detected. Moreover, as already pointed out in Remarks 3.2 and 3.4, the process of generating perceptual features could theoretically be stochastic.

The dark side of this convenient definition is that it clouds the involved computational mechanisms. These mechanisms can be quite complex, depending on the considered perceptual space. We now describe perceptual feature generators for some commonly used perceptual spaces.

Visual Spaces

Once a visual feature generator $\mathcal{G}_V$ has been fixed (cf. Definition 3.1), it is straightforward to derive a corresponding perceptual feature generator. It is indeed sufficient to collect the set of all the visual features that are generated across the input set of images:

$$\mathcal{G}_S(\{s_1, \dots, s_m\}) = \bigcup_{i=1}^{m} \mathcal{G}_V(s_i). \tag{4.10}$$

[2] The relation between this bank of features and our perceptual features are discussed in Section 4.3.

Note that in Chapter 5, a more complex strategy will be introduced. This generator will produce spatial combinations of visual features that are more discriminative than individual point features.

Continuous Spaces

When the perceptual space is continuous, a plausible way to extract thresholding features consists in considering each component of the input percepts as a threshold for this component:

$$\mathcal{G}_S(\{\boldsymbol{s_1}, \ldots, \boldsymbol{s_m}\}) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n} (s_{ij}, j). \tag{4.11}$$

Perceptual Spaces with Binary Numbers

In the case of perceptual spaces that consist of binary numbers, the perceptual feature generator can simply return the set of all possible bits. To this end, the generator first identifies the longest binary number in the furnished percepts, then returns all the indices below this threshold:

$$\mathcal{G}_S(\{s_1, \ldots, s_m\}) = \{0, \ldots, n\}, \tag{4.12}$$

where $n \in \mathbb{N}$ is the greatest integer such that:

$$(\exists i \in \{1, \ldots, n\})\ \mathcal{D}_S(s_i, n) = \mathbf{true}. \tag{4.13}$$

4.2 Learning Architecture

The key problem which arises when reinforcement learning is applied to large-scale (e.g. visual) or continuous problems is referred to as the *Bellman curse of dimensionality* [Bel57a]. Standard reinforcement learning algorithms such as those described in Chapter 2 directly use the "flat" perceptual space S. However, perceptual features allow to represent percepts in terms of properties they have, instead of the percepts themselves[3]. Thus, perceptual features can be used to add structure to the perceptual space.

The basic idea of the *Reinforcement Learning of Visual Classes*[4] (RLVC) algorithm consists in partitioning the perceptual space into a finite set of *perceptual classes* by focusing the attention of the agent on highly distinctive perceptual features [JP04, JP05c, JP05e]. Each percept is mapped to a perceptual class, and each perceptual class is in turn associated with a single symbol. This symbolic input

[3] Such feature-based representations of percepts are often called *intentional representations*.

[4] A better denomination for this algorithm would have been *Reinforcement Learning of Perceptual Classes*, as it can be as well applied to non-visual perceptual spaces. However, the algorithm was initially designed specifically for vision-for-action tasks. Thus, to keep the consistency with our publications, we have chosen to keep this original denomination.

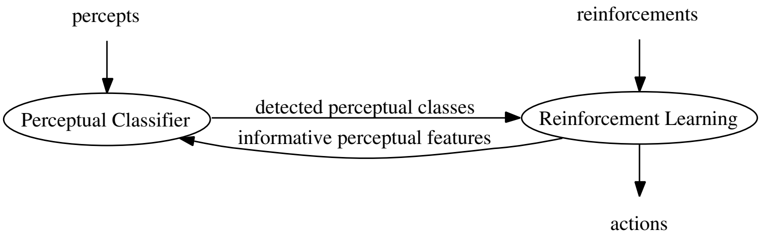


Figure 4.1: Sketch of the learning architecture of RLVC.

can then be used as the input to a classical, embedded RL algorithm, as shown in Figure 4.1. Indeed, this task-driven pre-processing step is intended to reduce the size of the input domain, thus enhancing the rate of convergence, the generalization capabilities as well as the robustness of RL to noise in complex perceptual domains. The central difficulty is the dynamic selection of the discriminative perceptual features. This selection process should group images that share homogeneous properties together in the same perceptual class.

To this end, RLVC consists of two simultaneous, interleaved processes: The incremental generation of a so-called *percept classifier* that maps a percept to the corresponding perceptual class, and the reinforcement learning of a mapping from the perceptual classes to the suitable actions.

Initially, the percept classifier knows about one perceptual class, so that any image is mapped to this class. Of course, this introduces a kind of *perceptual aliasing* (or *hidden state*) [WB91]: The optimal decisions cannot always be made, since percepts requiring different reactions are associated with the same class. The agent then isolates the aliased classes. Since there is no external supervisor, it can only rely on a statistical analysis of the earned reinforcements. For each detected aliased class, the agent dynamically selects a new perceptual feature that is *distinctive*, i.e. that best disambiguates the aliased percepts. The extracted feature is used to refine the classifier. This way, at each stage of the algorithm, the number of perceptual classes in the classifier grows.

New perceptual features are learned until perceptual aliasing vanishes. The resulting image classifier is finally used to control the system. Thus, RLVC selects a subset of highly relevant perceptual features in a fully closed-loop, task-driven learning process.

4.3 Related Work

The exploitation of perceptual features has already been actively addressed in the reinforcement learning literature.

4.3.1 Factored Representations of MDPs

In the context of dynamic programming, the use of perceptual features has led to the development of *factored representations*. In a *factored Markov decision process* [BDG00], the perceptual space is described using a finite set of intensional variables $\{x_1, \ldots, x_n\}$, where each variable x_i takes its values in a finite domain, and is termed a *factor* of the MDP. In our framework, a factor x_i would correspond to a perceptual feature that would have been selected through some suitable pre-processing. As the size of the perceptual space is exponential in the number of factors, it is impossible to directly represent the transition relation $\mathcal{T}$, the reinforcement signal $\mathcal{R}$, as well as the value functions or the policies.

Several compact representations based on the factors have therefore been defined. One way to encode the transition relation $\mathcal{T}$ is to use a separate *dynamic Bayesian network* [DK89, Mur02] for each action. This particular class of Bayesian networks is sometimes referred to as *two-stage temporal Bayesian networks*. Any factor that is not relevant to the action is not represented. If there are only a few relevant factors, this representation is much more compact than a flat representation of the transition probabilities. An alternative factored representation is based on the concept of *probabilistic STRIPS operators* [KHW95], that is a special kind of decision trees. As for the reinforcement signal $\mathcal{R}$, it is often encoded as a simple decision tree.

Even when the transition relation and the reinforcement signal have been represented compactly, the main challenge is to develop algorithms which can take these factored data structures and calculate similarly compact representations of the optimal policy and the value function. The *Structured Policy Iteration* algorithm is one of the few algorithms that maintain a factored representation throughout the entire computation [BDG00]. This algorithm encodes value functions and policies using decision trees. This technique allows the efficient solution of Markov decision problems in very large perceptual spaces, without explicit state enumeration.

The SPUDD algorithm is also designed to solve factored Markov decision problems [HSHB99]. SPUDD represents the value functions as *Algebraic Decision Diagrams* (ADD) [BFG+93a], that extend *Binary Decision Diagrams* (BDD) [Bry92][5]. This algorithm can optimally solve Markov decision problems whose perceptual space contains hundreds of millions of states. Unfortunately, ADD can only represent value functions that have only few hundreds of distinct values.

Further work about factored MDPs includes the doctoral dissertations by Sallans [Sal02] and Guestrin [Gue03]. All these techniques are defined in the framework of Markov decision processes. As a consequence, much of this work assumes that the parameters of the MDP are known, and usually that the factored representations of actions and reward function are also known, which is not the case in RLVC.

[5] Binary decision diagrams will be presented in Section 5.1.3.

4.3.2 Perceptual Aliasing

Standard reinforcement learning explicitly assumes that the agent is able to distinguish between the states of the environment using only its sensors: The perceptual space is said *fully observable*, and the right decisions can always be made on the basis of the percepts. If it is not the case (i.e. if the perceptual space is only *partially observable*), the agent cannot distinguish between any pair of states and thus will possibly not be able to take systematically the right decision. This phenomenon is known as the *perceptual aliasing* (or *hidden state*) problem, and is closely related to ours. Indeed, Section 4.2 has explained that the incremental selection of a set of perceptual features necessarily leads to a temporary perceptual aliasing, which RLVC tries to get rid of.

Early work in reinforcement learning has tackled this general problem in two distinct ways: Either the agent identifies and then avoids states where perceptual aliasing occurs (as in the Lion algorithm [WB91]), or it tries to build a short-term memory that will allow it to remove the ambiguities on its percepts (as in the *predictive distinctions* approach [Chr92]). Very sketchily, these two algorithms detect the presence of perceptual aliasing through an analysis of the sign of Q-learning updates (cf. Section 2.4.2). The possibility of managing a short-term memory has led to the development of the *Partially Observable Markov Decision Processes* (POMDP) theory [KLC98], in which the current state is a random variable of the percepts. Hasinoff's technical report [Has03] is an up-to-date reference about the perceptual aliasing issue.

Although these approaches are closely related to the perceptual aliasing RLVC temporarily introduces, they do not consider the exploitation of perceptual features. Indeed, they tackle a structural problem in a given control task, and, as such, they assume that perceptual aliasing cannot be removed. As a consequence, these approaches are orthogonal to our research interest, since the ambiguities RLVC generates can be removed by further refining the percept classifier. In fact, the techniques above tackle a *lack of information* inherent to the used sensors, whereas our goal is to handle a *surplus of information* related to the high redundancy of visual representations.

4.3.3 Adaptive Resolution in Finite Perceptual Spaces

RLVC performs an *adaptive* discretization of the perceptual space through an autonomous, task-driven selection of perceptual features. Work in RL that incrementally partitions a large (either discrete or continuous) perceptual space into a piecewise constant value function is usually referred to as *adaptive-resolution* techniques. Ideally, regions of the perceptual space with a high granularity should only be present where they are needed, while a lower resolution should be used elsewhere. RLVC is such an adaptive-resolution algorithm. We now review several adaptive-resolution methods that are suited for finite perceptual spaces.

The idea of adaptive-resolution techniques in reinforcement learning goes back to the G Algorithm [CK91], and has inspired the other approaches that are discussed below. The G Algorithm considers perceptual spaces that are made up of fixed-length binary numbers. It learns a decision tree that tests the presence of informative bits in the percepts. This algorithm uses a Student's t-test to determine if there is some bit b in the percepts that is mapped to a given leaf, such that the state-action utilities of states in which b is set are significantly different from the state-action utilities of states in which b is unset. If such a bit is found, the corresponding leaf is split. The process is repeated for each leaf. This method is able to learn compact representations, even though there is a large number of irrelevant bits in the percepts. Unfortunately, when a region is split, all the information associated with that region is lost, which makes for very slow learning. Concretely, the G Algorithm can solve a task whose perceptual space contains 2^{100} distinct percepts, which corresponds to the set of binary numbers with a length of 100 bits.

McCallum's *U-Tree* algorithm builds upon this idea by combining a "selective attention" mechanism inspired by the G Algorithm with a short-term memory that enables the agent to deal with partially observable environments [McC96]. Therefore, McCallum's algorithms are a keystone in reinforcement learning, as they unify the G Algorithm [CK91] with Chrisman's predictive distinctions [Chr92].

U-Tree incrementally grows a decision tree through Kolmogorov-Smirnov tests. It has succeeded at learning behaviors in a driving simulator. In this simulator, a percept consists in a set of 8 discrete variables whose variation domains contain between 2 and 6 values, leading to a perceptual space with $2,592$ possible percepts. Thus, the size of the perceptual percept is much smaller than a visual space. However, this task is difficult because the "physical" state space is only partially observable through the perceptual space: The driving task contains $21,216$ physical states, which means that several physical states requiring different reactions can be mapped to the same percept through the sensors of the agent. U-Tree resolves such ambiguities on the percepts by testing the presence of perceptual features in the percepts that have been encountered previously in the history of the system. To this end, U-Tree manages a short-term memory. In this dissertation, we do not consider partially observable environments. Our challenge is rather to deal with huge visual spaces, without hand-tuned pre-processing, which is in itself a difficult, novel research direction.

More recently, Porta and Celaya have also considered adaptive-resolution algorithms for complex finite perceptual spaces [PC05]. They propose to apply reinforcement learning only on a subset of the perceptual input. In their framework, a perceptual input is a vector of Booleans with a fixed size that contains the binary answers of a set of feature detectors. This concept is very similar to our notion of perceptual features. However, their algorithms handle complex *action* spaces simultaneously with complex perceptual spaces. As discussed in Chapter 7, this is an important contribution, as RL algorithms are also highly sensitive to the number of possible actions. Although RLVC is limited to discrete, small-sized action spaces, this requirement will be lifted in the generalized version of RLVC that will be called *Reinforcement Learning of Joint Classes*.

4.3.4 Adaptive Resolution in Continuous Perceptual Spaces

It is important to notice that all the methods for adaptive resolution in large-scale, finite perceptual spaces use a fixed set of perceptual features that is hard-wired. This has to be distinguished from RLVC that samples perceptual features from a possibly *infinite feature space* (e.g. the visual feature space is infinite), and that makes no prior assumptions about the maximum number of useful features. From this point of view, RLVC is closer to adaptive-resolution techniques for continuous perceptual spaces. Indeed, these techniques can dynamically select new relevant features from a whole continuum.

The very first research in adaptive-resolution algorithms for continuous perceptual spaces is the DARLING algorithm [Sal93]. This algorithm, just like all the current algorithms for continuous adaptive resolution, splits the perceptual space using thresholds, as described in Section 4.1.2. For this purpose, DARLING builds a hybrid decision tree that assigns a label to each point in the perceptual space. This is a fully on-line and incremental algorithm that is equipped with a *forgetting mechanism* that deletes outdated interactions. DARLING is however limited to binary reinforcement signals, and it only takes immediate reinforcements into account, so that DARLING is much closer to supervised learning than to reinforcement learning.

The *Parti-Game* algorithm [MA95] produces goal-directed behaviors in continuous perceptual spaces. Parti-Game also splits regions where it deems it important, but the approach and assumptions are significantly different. Parti-Game indeed assumes that the agent possesses a greedy controller that can move it towards any desired state. There is no guarantee that the greedy controller will succeed, but the agent is warned when the greedy controller becomes stuck against some obstacle. Parti-Game assumes that the dynamics of the world are deterministic and that the goal state is known. The objective is to produce behavior that takes the agent to the goal without becoming stuck, which is different from the objective of reinforcement learning where the agent must maximize its expected discounted return. Parti-Game uses a game-theoretic approach to decide which regions to split. Moore and Atkeson show that Parti-Game can learn competent behavior in a variety of continuous domains. Unfortunately, the approach is currently limited to deterministic domains where the agent has a greedy controller. Moreover, this algorithm searches for *any* solution to a given task, and does not try to find the optimal one.

The *Continuous U-Tree* algorithm is an extension of U-Tree that is adapted to continuous perceptual spaces [UV98]. Just like DARLING, Continuous U-Tree incrementally builds a decision tree that splits the perceptual space in a finite set of hypercubes, by testing thresholds. Kolmogorov-Smirnov and sum-of-squared-errors are used to determine when to split a node in the decision tree. Pyeatt and Howe analyze the performance of several splitting criteria for a variation of Continuous U-Tree [PH01]. They conclude that Student's t-test leads to the best performance. Continuous U-Tree has been extended to Semi-Markov Decision Processes (cf. Section 2.2) by Uther in the *T-Tree* algorithm [Uth02].

Munos and Moore have proposed *Variable Resolution Grids* [MM02]. Their algorithm assumes that the perceptual space is a compact subset of Euclidean space, and begins with a coarse, grid-based discretization of the state space. In contrast with the other abstract algorithms in this section, the value function and policy vary linearly within each region. Munos and Moore use Kuhn triangulation as an efficient way to interpolate the value function within regions. The algorithm refines its approximation by splitting cells according to a splitting criterion. Munos and Moore explore several local heuristic measures of the importance of splitting a cell including the average of corner-value differences, the variance of corner-value differences, and policy disagreement. They also explore global heuristic measures involving the influence and variance of the approximated system. The influence is a measure of non-local dependencies in the value function, and variance is an estimate of the error in the value function due to the grid approximation. Variable Resolution Grids are probably the most advanced adaptive-resolution algorithm available so far.

4.3.5 Discussion

To summarize, several algorithms that are similar in spirit to RLVC have been proposed over the years. Nevertheless, our work appears to be the first that can learn direct image-to-action mappings through reinforcement learning. Indeed, none of the methods above combines all the following desirable properties of RLVC:

1. It is defined in the reinforcement learning framework, which is important from the point of view of purposive vision (cf. the introductory chapter);

2. The set of relevant perceptual features is not chosen a priori by hand, as the selection process is fully automatic and does not require any human intervention;

3. Visual perceptual spaces are explicitly considered through appearance-based visual features; and

4. The highly informative perceptual features can be drawn out of a possibly infinite set.

These advantages of RLVC are essentially due to the fact that the candidate perceptual features are not selected only because they are informative: They are also *ranked* according to some information-theoretic measure that is inspired by the learning of decision trees [BFS84]. Such a ranking is required, as vision-for-action tasks induce a large number of visual features (a typical image contains about a thousand of them). This kind of criterion that ranks perceptual features, though already considered in Variable Resolution Grids [MM02], seems to be new in discrete perceptual spaces. This gives to RLVC a competitive advantage over previous adaptive-resolution algorithms in finite perceptual spaces, and this is the main feature that allows our algorithms to learn image-to-action mappings.

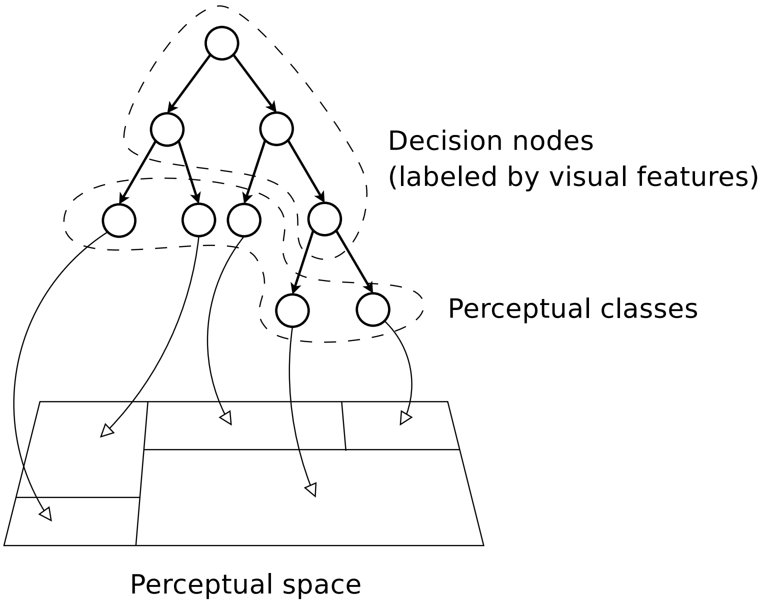


Figure 4.2: A percept classifier and its effects on the perceptual space. (This illustration is strongly inspired by Pyeatt and Howe [PH01].)

4.4 Adaptive Discretization of the Perceptual Space

As discussed in Section 4.2, we propose to insert a percept classifier before the RL algorithm. This classifier maps the stimuli to a set of perceptual classes, by focusing the attention of the agent on highly distinctive perceptual features that are extracted through a perceptual feature generator $\mathcal{G}_S$. RLVC assumes the finiteness of the action space: $A = \{a_1, \dots, a_m\}$. However, the perceptual space S can possibly be infinite.

The percept classifier is iteratively refined. Because of this incremental process, a natural way to implement the percept classifiers is to use binary decision trees. Each of their internal nodes is labeled by the perceptual feature, the presence of which is to be tested in that node. The n leaves of the trees define a set of n perceptual classes, which is hopefully much smaller than the original perceptual space S, and upon which it is possible to apply directly any usual RL algorithm. To classify a percept, the system starts at the root node, then progresses down the tree according to the result of the feature detector $\mathcal{D}_S$ for each perceptual feature found during the descent, until reaching a leaf. This process is illustrated in Figure 4.2.

To summarize, RLVC builds a sequence $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \dots$ of growing decision trees, in a sequence of attempts to remove perceptual aliasing. At any stage k, the classifier $\mathcal{C}_k$ partitions the perceptual space S into a finite number m_k of perceptual classes:

Definition 4.6. A *percept classifier* $\mathcal{C}_k : S \mapsto S_k$ is a mapping from the percepts to

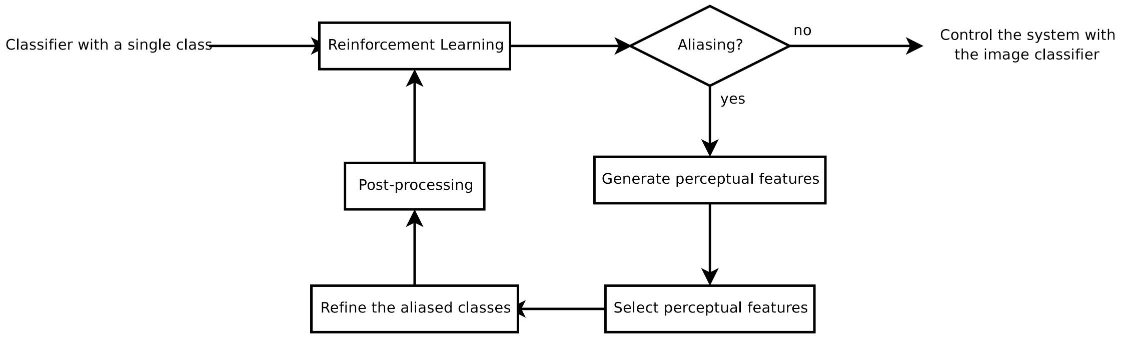

Classifier with a single class
Reinforcement Learning
Aliasing?
no
Control the system with the image classifier
yes
Generate perceptual features
Post-processing
Select perceptual features
Refine the aliased classes

Figure 4.3: The different components of the RLVC algorithm.

a set of perceptual classes S_k that is defined as:

$$S_k = \left\{ c_1^{(k)}, \dots, c_{m_k}^{(k)} \right\}. \tag{4.14}$$

The initial classifier $\mathcal{C}_0$ maps all of its input percepts to a single perceptual class $c_1^{(0)}$.

The components of RLVC are depicted in Figure 4.3. An in-depth discussion of each of these components will be given in the next sections. We first briefly review each of them:

RL algorithm: For each classifier in the sequence, an arbitrary, standard RL algorithm is applied. This provides information such as the optimal state-action function, the optimal value function or the optimal policy that are induced by the current classifier $\mathcal{C}_k$. For the purpose of these computations, either new interactions can be acquired, or a database of previously collected interactions can be exploited. This component is covered in Section 4.4.1.

Note that in all the experiments that are presented in this dissertation, model-based RL algorithms have been applied to static databases of interactions. These databases have been collected using a fully randomized exploration strategy (cf. Section 2.4.2).

Aliasing detector: In RLVC, the embedded reinforcement learning algorithm does not perceive the system directly through its sensors, but rather through a percept classifier. As a consequence, until the agent has learned the perceptual classes that are required to complete its task, the input space is only partially observable from the point of view of the embedded RL algorithm.

Therefore, at regular intervals, an aliasing detector extracts the classes in which perceptual aliasing occurs, through an analysis of the Bellman residuals (cf. Section 2.4.2). Indeed, as explained in Section 4.4.2, tight relations exist between perceptual aliasing and Bellman residuals. If no aliased class is detected, RLVC stops. This periodic check testing whether to add additional distinctions to the leaves of the tree is also present in U-Tree [McC96] and Continuous U-Tree [UV98].

Perceptual feature generator: After applying the RL algorithm, a database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$ is available. The perceptual feature generator $\mathcal{G}_S$ produces a set $F_i^{(k)}$ of candidate perceptual features for each aliased class c. This is done by applying $\mathcal{G}_S$ on the set of percepts that are mapped to the aliased classes. The features that will be used to refine a classifier will be chosen among this set of candidates.

Feature selector: Once the set of candidate features $F_i^{(k)}$ is built, this component selects the perceptual feature $f^* \in F_i^{(k)}$ that best reduces the perceptual aliasing in the perceptual class c. This step is further described in Section 4.4.3.

Classifier refinement: The leaves that correspond to the aliased classes are replaced by an internal node testing the presence or absence of the selected perceptual features.

Post-processing: This optional component is invoked after every refinement, and corresponds to techniques for reducing overfitting effects. A suitable post-processing procedure will be detailed in Chapter 5.

The resulting general outline of RLVC is described in Algorithm 4.1. The following sections describe the remaining algorithms, namely `aliased`, `selector` and `post-process`.

4.4.1 Mapping an MDP through a Percept Classifier

We now discuss how aliasing can be detected in a classifier $\mathcal{C}_k$. Formally, any percept classifier $\mathcal{C}_k$ converts a sequence of N interactions $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$, to a *mapped sequence* of N quadruples:

$$\langle \mathcal{C}_k(s_t), a_t, r_{t+1}, \mathcal{C}_k(s_{t+1})\rangle \in S_k \times A \times \mathbb{R} \times S_k, \tag{4.15}$$

where S_k is the set of perceptual classes that are known to $\mathcal{C}_k$. Each of these quadruples corresponds to an interaction in a so-called mapped MDP $\mathcal{M}_k$, whose set of states corresponds to the set of perceptual classes that are known to $\mathcal{C}_k$:

Definition 4.7. Let $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$ be a sequence of interactions that comes from an exploration of the MDP $\mathcal{M}$. Let also $\mathcal{C}_k$ be a percept classifier that is generated by RLVC. The *mapped MDP* $\mathcal{M}_k$ that is derived from this sequence of interactions, is the quadruple:

$$\langle S_k, A, \mathcal{T}_k, \mathcal{R}_k\rangle, \tag{4.16}$$

where $\mathcal{T}_k$ and $\mathcal{R}_k$ are constructed from the relative frequencies that appear in the mapped sequence $\langle \mathcal{C}_k(s_t), a_t, r_{t+1}, \mathcal{C}_k(s_{t+1})\rangle$. Precisely, $\mathcal{T}_k$ (resp. $\mathcal{R}_k$) are derived from the mapped sequence through Equations 2.40 (resp. 2.41).

Each mapped MDP $\mathcal{M}_k$ is characterized by an optimal state-action value function on the domain $S_k \times A$ that will be denoted $\hat{Q}_k^*$. RLVC applies the embedded RL

Algorithm 4.1 — General structure of RLVC

```
$k \leftarrow 0$
$m_k \leftarrow 1$
$\mathcal{C}_k \leftarrow$ binary decision tree with one leaf
repeat
    Collect $N$ interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$
    Apply some RL algorithm on the sequence that is mapped through $\mathcal{C}_k$
    $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_k$
    for $i \leftarrow 1$ to $m_k$ do
        $c \leftarrow c_i^{(k)}$
        if aliased$(c)$ then
            $F \leftarrow \mathcal{G}_S\left(\{s_t \mid \mathcal{C}_k(s_t) = c\}\right)$
            $f^* \leftarrow$ selector$(c, F)$
            if $f^* \neq \bot$ then
                $\mathcal{C}_{k+1} \leftarrow \mathcal{C}_{k+1}$, where the perceptual class $c$ is refined by a test on $f^*$
                $m_{k+1} \leftarrow m_{k+1} + 1$
            end if
        end if
    end for
    $k \leftarrow k + 1$
    post-process$(\mathcal{C}_k)$
until $\mathcal{C}_k = \mathcal{C}_{k-1}$
```

algorithm to the mapped sequence of interactions to obtain this optimal state-action value function. In turn, the function $\hat{Q}_k^*$ induces another state-action value function Q_k^* on the initial domain $S \times A$ through the relation:

$$Q_k^*(s,a) = \hat{Q}_k^*\left(\mathcal{C}_k(s), a\right). \tag{4.17}$$

Computing $\hat{Q}_k^*$ can be difficult: In general, there may exist no MDP defined on the state space S_k and on the action space A that can generate a given mapped sequence, given that the latter is not necessarily Markovian anymore. Thus, if some RL algorithm is run on the mapped sequence, it might not converge toward $\hat{Q}_k^*$, or not even converge at all. However, when applied on the mapped MDP $\mathcal{M}_k$, any model-based RL method (cf. Section 2.4.1) can be used to compute $\hat{Q}_k^*$. Under some conditions, Q-learning also converges to the optimal state-action value function of the mapped MDP [SJJ95].

4.4.2 Measuring Aliasing

In the absence of aliasing, the agent could perform optimally, and the state-action value function Q_k^*, as defined by Equation 4.17, would correspond to the optimal state-action value function Q^*, according to Bellman theorem that states the uniqueness of the optimal Q function (cf. Section 2.2.3). By Equation 2.29, the function

$$B_k(s,a) = (HQ_k^*)(s,a) - Q_k^*(s,a), \tag{4.18}$$

where H is the Bellman backup operator for the MDP $\mathcal{M}$, is therefore a measure of the aliasing induced by the image classifier $\mathcal{C}_k$ with respect to the percept $s \in S$ and to the action $a \in A$.

The basic idea behind RLVC is to refine the perceptual classes containing a percept $s \in S$ such that $B_k(s,a)$ is non-zero for some action $a \in A$. To evaluate the function $B_k(s,a)$, all the learning agent has at its disposal is a database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$. Consider a time stamp t in this database. According to Equation 4.18 and to the definition of the Bellman backup operator H, the value of B_k for the state-action pair (s_t, a_t) equals

$$B_k(s_t,a_t) = \mathcal{R}(s_t,a_t) + \gamma \sum_{s' \in S} \mathcal{T}(s_t,a_t,s') \max_{a' \in A} Q_k^*(s',a') - Q_k^*(s_t,a_t). \tag{4.19}$$

Unfortunately, the learning agent does not have access to the transition probabilities $\mathcal{T}$ and to the reinforcement function $\mathcal{R}$ of the MDP $\mathcal{M}$ modeling the environment. Therefore, Equation 4.19 cannot be directly evaluated by the agent. A similar issue arises in the Q-learning [Wat89] and the Fitted Q Iteration [EGW05] algorithms. As discussed in Section 2.4.2 in the context of Q-learning, these algorithms solve this problem by considering the stochastic version of the time difference that is described by Equation 4.19. Following the same line of reasoning as in Section 2.4.2, the value

$$\sum_{s' \in S} \mathcal{T}(s_t,a_t,s') \max_{a' \in A} Q_k^*(s',a') \tag{4.20}$$

can be estimated as

$$\max_{a' \in A} Q_k^*(s',a'), \tag{4.21}$$

if the successor s' is chosen with probability $\mathcal{T}(s_t,a_t,s')$. But following the transitions of the environment ensures making a transition from s_t to s_{t+1} with probability $\mathcal{T}(s_t,a_t,s_{t+1})$. Thus

$$\begin{aligned} \Delta_t &= r_{t+1} + \gamma \max_{a' \in A} Q_k^*(s_{t+1},a') - Q_k^*(s_t,a_t) && (4.22)\\ &= r_{t+1} + \gamma \max_{a' \in A} \hat{Q}_k^*\left(\mathcal{C}_k\left(s_{t+1}\right),a'\right) - \hat{Q}_k^*(\mathcal{C}_k(s_t),a) && (4.23)\end{aligned}$$

is an unbiased estimate of Equation 4.19. The value Δ_t is the Bellman residual that is used to iteratively update the estimate of the optimal state-action value function in Q-learning (cf. Section 2.4.2).

As previously discussed when deriving Q-learning, if the system is deterministic, in the absence of perceptual aliasing, and after convergence to the optimal state-action value function, these Bellman residuals are equal to zero. Therefore, a nonzero Δ_t potentially indicates the presence of perceptual aliasing in the perceptual class $c_t = \mathcal{C}_k(s_t)$ with respect to action a_t.

Our criterion for detecting the aliased classes is now described. First, the $\hat{Q}_k^*$ function is computed. Secondly, let c be a perceptual class that belongs to the

Algorithm 4.2 — Aliasing Criterion of RLVC

```
aliased$(c)$ :-
  for $a \in A$ do
    $\Delta \leftarrow \{\Delta_t \mid \mathcal{C}_k(s_t) = c \text{ and } a_t = a\}$
    if $\sigma^2(\Delta) > \tau$ then
      return true
    end if
  end for
  return false
```

percept classifier $\mathcal{C}_k$, and let $a \in A$ be an action. Consider the following set of time stamps:

$$T\,(c, a) = \{t \mid \mathcal{C}_k(s_t) = c \text{ and } a_t = a\}. \tag{4.24}$$

Each element in this set indexes an interaction that is simultaneously related to the class c and to the action a. We assert the presence of aliasing in the perceptual class c with respect to an action $a \in A$ if the set of Bellman residuals

$$\{\Delta_t \mid t \in T\,(c, a)\} \tag{4.25}$$

has a variance that exceeds a given threshold $\tau \in \mathbb{R}_0^+$. This is summarized in Algorithm 4.2, where $\sigma^2(\cdot)$ denotes the variance of a set of samples.

4.4.3 Selecting Distinctive Perceptual Features

Once aliasing has been detected in some perceptual class $c \in S_k$ with respect to an action a, we need to select a perceptual feature that best explains the variations in the set of Δ_t values corresponding to c and a. This perceptual feature is to be chosen among the set of candidates $F_i^{(k)}$. This is a regression problem, for which we suggest an adaptation of a popular splitting rule used in the CART algorithm for building regression trees [BFS84].

In CART, variance is used as an impurity indicator: The split that is selected to refine a particular node is the one that leads to the greatest reduction in the sum of the squared differences between the response values for the learning samples corresponding to the node and their mean. More formally, if the learning samples are $L = \{\langle \boldsymbol{x_i}, y_i \rangle\}$, then the feature that is selected is:

$$f^* = \underset{v \in F_i^{(k)}}{\operatorname{argmin}} \left(p_\oplus^v \cdot \sigma^2 \left(L_\oplus^v \right) + p_\ominus^v \cdot \sigma^2 \left(L_\ominus^v \right) \right), \tag{4.26}$$

where $p_\oplus^v$ (resp. $p_\ominus^v$) is the proportion of samples $\langle \boldsymbol{x_i}, y_i \rangle$ that exhibit (resp. do not exhibit) the feature v in $\boldsymbol{x_i}$, and where $L_\oplus^v$ (resp. $L_\ominus^v$) is the set of outputs y_i corresponding to samples that exhibit (resp. do not exhibit) the feature v in $\boldsymbol{x_i}$. This idea can be directly transferred in our framework, if the set of $\boldsymbol{x_i}$ corresponds to the set of aliased percepts s_t, and if the set of y_i corresponds to the set of Bellman residuals Δ_t. This is written explicitly in Algorithm 4.3.

Algorithm 4.3 — Feature selection process

```
selector $(c, F)$ :−
  $f^* \leftarrow \perp$          {Best feature found so far}
  $r^* \leftarrow +\infty$        {Variance reduction induced by $f^*$}
  for $a \in A$ do
    $T \leftarrow \{t \mid \mathcal{C}_k(s_t) = c$ and $a_t = a\}$
    for perceptual features $f \in F$ do
      $S_\oplus \leftarrow \{\Delta_t \mid t \in T$ and $\mathcal{D}_S(s_t, f) =$ true$\}$
      $S_\ominus \leftarrow \{\Delta_t \mid t \in T$ and $\mathcal{D}_S(s_t, f) =$ false$\}$
      $p_\oplus \leftarrow |S_\oplus| / |T|$
      $p_\ominus \leftarrow |S_\ominus| / |T|$
      $r \leftarrow p_\oplus \cdot \sigma^2(S_\oplus) + p_\ominus \cdot \sigma^2(S_\ominus)$
      if $r < r^*$ and the distributions $(S_\oplus, S_\ominus)$ are significantly different then
        $f^* \leftarrow f$
        $r^* \leftarrow r$
      end if
    end for
  end for
  return $f^*$
```

Our algorithms exploit the stochastic version of Bellman residuals. Of course, real environments are in general non-deterministic, and thus generate variations in Bellman residuals that are not a consequence of perceptual aliasing. RLVC can be made somewhat robust to such a variability by introducing a statistical hypothesis test: For each candidate feature, a Student's t-test is used to decide whether the two sub-distributions that the feature induces are significantly different.

This is precisely the approach that is used when generating binary decision trees: Whenever a leaf node should be split, the components that have an influence on the outcome variable are selected, then the most relevant component among these candidates is used to refine the node. The choice of Student's t-test to detect which variables have an influence, which originates in the G Algorithm [CK91], is motivated by Pyeatt and Howe's comparison [PH01] as well as by the CART algorithm for learning regression trees [BFS84]. As for the variance reduction score, it is employed in the learning of regression trees through CART [BFS84].[6]

4.5 The Binary Gridworld Application

In this chapter, RLVC will be evaluated on several navigation tasks with increasing complexity. We first consider the application of RLVC for to solve of *discrete* 2D

[6] Note that the general architecture of RLVC is independent of this choice: Although we were inspired by the learning of CART decision trees, any other algorithm for learning regression trees can be inserted as the inner loop of Algorithm 4.3. For example, in our previous papers, we have used a splitting rule that is borrowed from the building of classification trees [Qui93, JP05c].

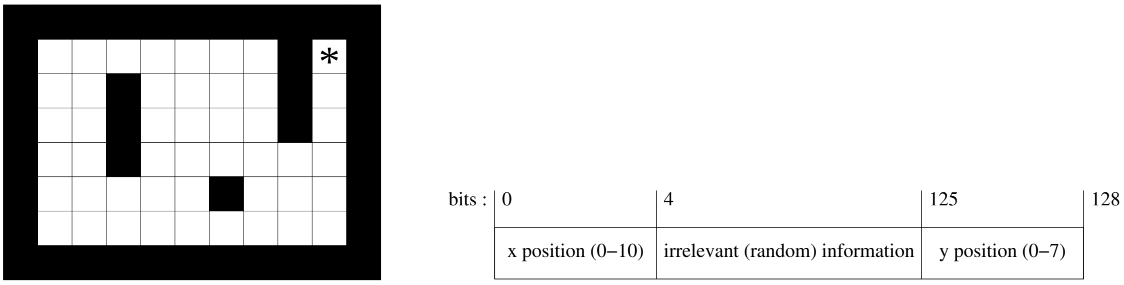


Figure 4.4: On the left, Sutton's Gridworld [Sut90]. Filled squares are walls, and the exit is indicated by an asterisk. On the right, a diagram describing the percepts of the agent, that are binary numbers of 128 bits. The x and y positions are encoded as binary numbers of respectively 4 and 3 bits, and are concatenated with a random binary number.

mazes that are constituted of walls and empty cells. There is one exit in the mazes. In each cell, the agent has four possible actions: Go up, right, down, or left. If a move would take the agent into a wall, its location is not changed. If a move takes it into the exit, the agent is randomly teleported elsewhere in the maze. The agent earns a reward of 100 when the exit is reached, and a penalty of -1 for any other move. Note that the agent is faced with the delayed reward problem.

This task is directly inspired by Sutton's so-called *Gridworld task* [Sut90], with the major exception that our agent does not have a direct access to its (x, y) position in the maze. Rather, the position is implicitly encoded in the percepts: In the current section, the percepts will be binary numbers that contain the binary values of x and y; at a later time, the position will be encoded as images (cf. Section 4.6.2) to prove the capability of RLVC to deal with vision-for-action tasks. In this first experiment, we have used the original Gridworld topology, which is depicted at the left of Figure 4.4. The sensors of the agent return a binary number, the structure of which is shown on the right of the same figure. Therefore, the perceptual space is $S = \mathcal{B}^{128}$. This is a perceptual domain that has a complexity that is quite similar to that of the experimental setup used to evaluate the G Algorithm [CK91].

Because the percepts consist of a binary number, the perceptual feature space is defined as the set of bits $F = \mathbb{N}$, and the perceptual feature detector and generator are defined according to Sections 4.1.2 and 4.1.3 for perceptual spaces with binary numbers. As a reminder, the perceptual feature detector consists in testing whether a given bit is set in a binary number, and the perceptual feature generator returns the list of the non-constant bits in a given set of binary numbers. RLVC has been run on a static database of $5,000$ interactions that has been collected through a fully randomized exploration policy.

To achieve its task, the agent has to focus its attention on the bits encoding x and y; the other bits are irrelevant to the task because they are random. We have noticed that this is indeed the case: The built classifier only uses the bits 0, 1, 2, 3, 125, 126 and 127. The obtained percept classifier is depicted in Figure 4.5, and the classes as well as the optimal percept-to-action mapping that the classifier induce are reported

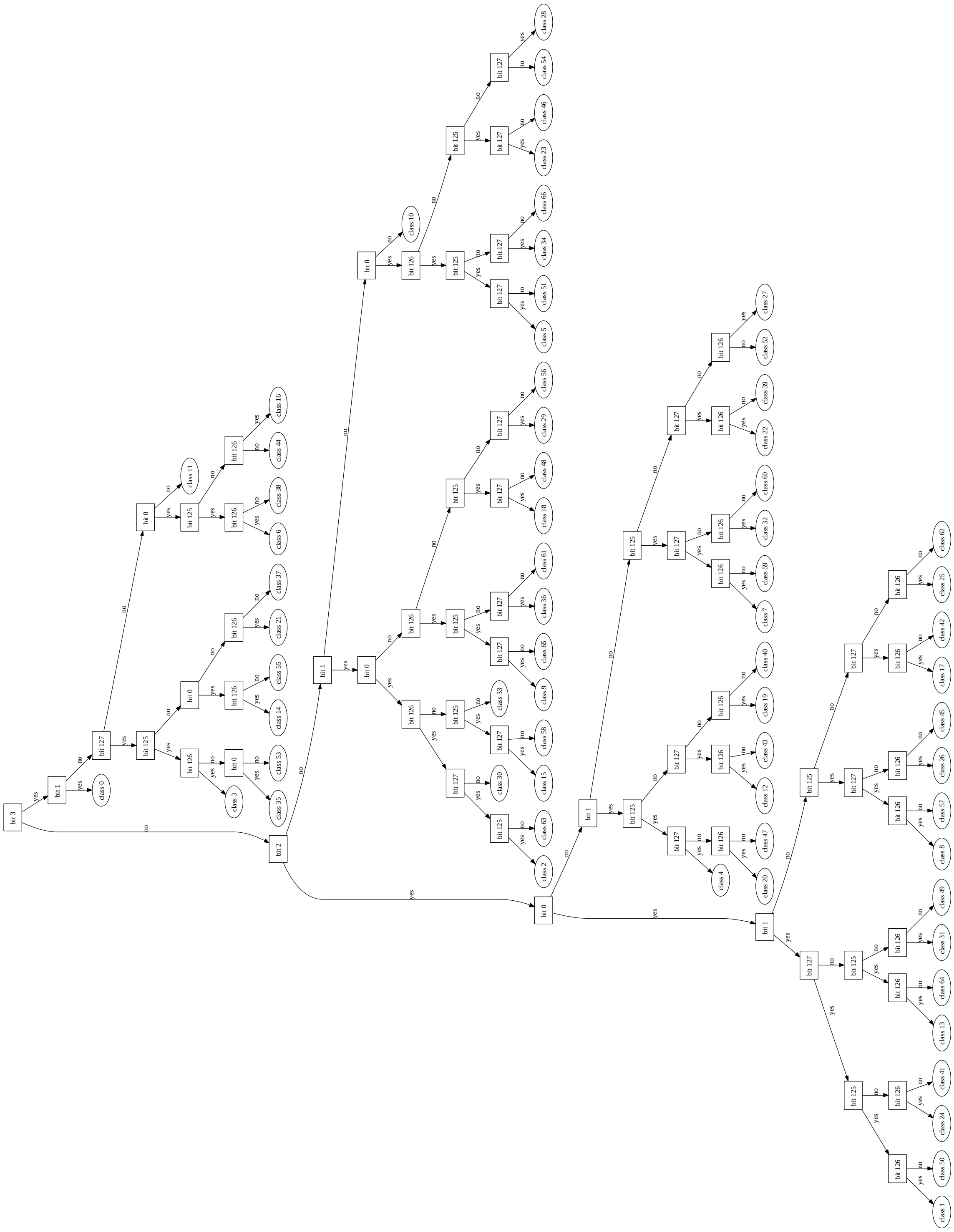
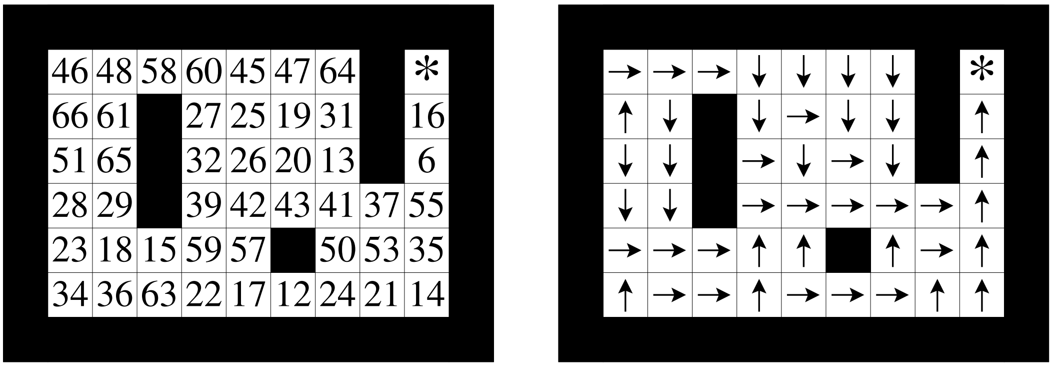


Figure 4.5: The percept classifier $\mathcal{C}_k$ that is obtained at the end of the RLVC process. Note that this decision tree contains no test on the random bits present in the percepts.



46	48	58	60	45	47	64	■	*
66	61	■	27	25	19	31	■	16
51	65	■	32	26	20	13	■	6
28	29	■	39	42	43	41	37	55
23	18	15	59	57	■	50	53	35
34	36	63	22	17	12	24	21	14

→	→	→	↓	↓	↓	↓	■	*
↑	↓	■	↓	→	↓	↓	■	↑
↓	↓	■	→	↓	→	↓	■	↑
↓	↓	■	→	→	→	→	→	↑
→	→	→	↑	↑	■	↑	→	↑
↑	→	→	↑	→	→	→	↑	↑

Figure 4.6: On the left, the classes that are induced by the percept classifier that is generated through RLVC. On the right, the optimal percept-to-action mapping that is built when reinforcement learning is applied through this classifier.

in Figure 4.6. It can easily be checked that the built policy is optimal. After k has reached the value 15 (which roughly corresponds to the diameter of the maze) no further split was produced. Note however that this value can vary depending on the collected database of interactions. The number of generated perceptual classes was 67, which is slightly less than the total number of cells ($11 \times 8 = 88$) because there is no need to make distinctions between the walled-up cells. The full learning time was 5 seconds on a 3GHz Pentium IV.

It is important to notice that the classification rule is obtained without pretreatment, nor human intervention. The agent is initially not aware of which bits are important, it just knows that it should consider bit features. Moreover, the interest of using features is clear in this application: A direct tabular representation of the Q function would have $2^{128} \times 4$ cells (one for each possible pair with a binary number and an action).

4.6 Visual Applications

The behavior of RLVC is now investigated on several vision-for-action tasks. The tasks considered are navigation problems in which the agent is equipped with visual sensors. The goal of all these tasks consists in escaping from a maze as quickly as possible. Local-appearance visual features (cf. Definition 3.6) are used to solve these problems.

4.6.1 Details of Implementation

We first describe several details of implementation that may prove to be useful when RLVC is used to solve vision-for-action tasks.

Limitation of the Number of Splits

Using Algorithm 4.1 directly might potentially result in the creation of a large number of perceptual classes, and hereby in overfitting. As a consequence, it is very useful in practice to bound the number of perceptual classes that can be refined at each stage of the algorithm, since splitting *one* visual class potentially has an impact on the Bellman residuals of *all* the perceptual classes. In practice, we first try to split the classes that have the most samples before considering the others, since there is more statistical evidence of variance reduction for the former. From an algorithmic point of view, this means that the loop over i in Algorithm 4.1 favors the perceptual classes c such that

$$\sum_{a \in A} T(c, a) \tag{4.27}$$

is the largest. In our tests, we systematically apply this heuristic and we limit to 5 the number of perceptual classes that are refined for a given $\mathcal{C}_k$.

Mahalanobis Distance

Throughout the experiments of this dissertation, the Mahalanobis distance will be used as the metric on the visual feature space V. The Mahalanobis distance is a generalization of the Euclidean distance. It takes the correlation between the components of the visual features into account. A correlation means that there exists an association between a pair of components. The Mahalanobis distance makes uniform the influence of such associated components by an anisotropic rescaling. From a geometrical perspective, whereas the locus of the visual features at a fixed Euclidean distance from the origin of the axes is a spheroid, the same locus is a rotated ellipsoid under the Mahalanobis distance. The axes of the ellipsoid define the rescaling factors, and the rotation allows to reflect the correlations between the components:

Definition 4.8. Let $\Sigma \in \mathbb{R}^{n \times n}$ be a covariance matrix for the visual feature space $V = \mathbb{R}^n$. Let $\boldsymbol{x}, \boldsymbol{y} \in V$ two visual features. The Mahalanobis distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as:

$$d_\Sigma(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{y})}. \tag{4.28}$$

If Σ is the identity matrix, the Mahalanobis distance corresponds to the Euclidean distance.

The covariance matrix Σ can be estimated from a finite set of representative images $\{s_1, \ldots, s_n\} \subset S$. The idea consists in collecting all the visual features that are extracted by the visual feature generator:

$$C = \bigcup_{i=1}^{n} \mathcal{G}_V(s_i); \tag{4.29}$$

then the covariance matrix Σ of C is computed as follows:

$$\Sigma = \begin{bmatrix} \mathrm{cov}(v_1, v_1) & \mathrm{cov}(v_1, v_2) & \dots & \mathrm{cov}(v_1, v_n) \\ \mathrm{cov}(v_2, v_1) & \mathrm{cov}(v_2, v_2) & \dots & \mathrm{cov}(v_2, v_n) \\ \vdots & & & \vdots \\ \mathrm{cov}(v_n, v_1) & \mathrm{cov}(v_n, v_2) & \dots & \mathrm{cov}(v_n, v_n) \end{bmatrix}, \tag{4.30}$$

where $\mathrm{cov}(v_i, v_j)$ denotes the covariance of the components v_i and v_j that can be computed as:

$$\mathrm{cov}(v_i, v_j) = \sum_{\boldsymbol{v} \in C} \frac{(v_i - \bar{v}_i)(v_j - \bar{v}_j)}{|C|}, \tag{4.31}$$

where $\bar{v}_i$ denotes the mean of the ith component in the collection C, and $|C|$ is the number of visual features in the collection. In practice, computing Equation 4.28 is costly, but it is possible to reduce this formula to a simple Euclidean distance through a careful base change that makes use of the diagonalization of matrix Σ^{-1}. This technique is detailed by Schmid and Mohr [SM97].

4.6.2 Illustration on Visual Gridworld Tasks

We now apply RLVC to solve a vision-for-action task that is an adaptation of the experimental setup of Section 4.5. The navigation rules are kept identical, but the position is now implicitly encoded in the percepts by an *image* instead of a binary number. In each cell, a different object is stored under a transparent glass, and the sensors of the agent return a color picture of the object underneath. For this benchmark, we have used the pictures from the COIL-100 database [NNM96]. In our setup, the local-appearance feature generator consists in the combination of Harris color points of interest with color differential invariants. We have made experiments for this task under different configurations:

Small visual Gridworld: The topology for this first experiment is depicted in Figure 4.7. Figure 4.8 shows the obtained results. It can easily be seen that the policy built using the last classifier is indeed optimal for the task, since the algorithm succeeds at distinguishing between all the 7 visual inputs. There were 120 distinct visual features, but RLVC has only selected 6 features. The algorithm stopped once k reached the value of 6.

Large visual Gridworld: In a second experiment, we have used Sutton's original Gridworld topology [Sut90], which is depicted in Figure 4.9. Here again, RLVC managed in $k = 45$ steps to build a classifier that allows the agent to distinguish between all the states, and therefore to optimally solve its task. This classifier is too large to be visualized in this dissertation. However, it is very interesting to note that RLVC has selected only 46 different features among the 1080 possible ones. In fact, the algorithm has produced 47 perceptual classes, each of these corresponding exactly to one cell, allowing it to produce here again the optimal policy.

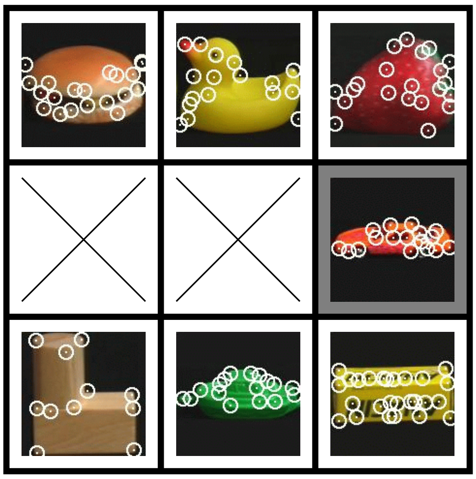
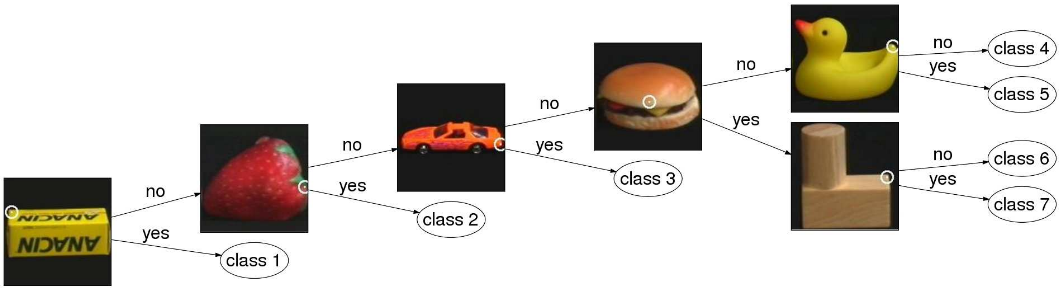


Figure 4.7: Small visual Gridworld topology. Cells with a cross are walls, and the exit is indicated by a gray background. Empty cells are labeled by a picture, in which circles indicate the interest points that are detected by the Color Harris detector.



(a)

6	5	2
☒	☒	3
7	4	1

(b)

→	→	↓
☒	☒	*
→	→	↑

(c)

Figure 4.8: Resolution of the small visual Gridworld by RLVC: (a) The final classifier $\mathcal{C}_k$ that tests the presence of the circled local-appearance features, (b) the label of the perceptual class that is assigned to each empty cell by $\mathcal{C}_k$, and (c) the computed optimal policy for this classification, i.e. $\operatorname{argmax}_{a \in A} Q_k^*(s, a)$.

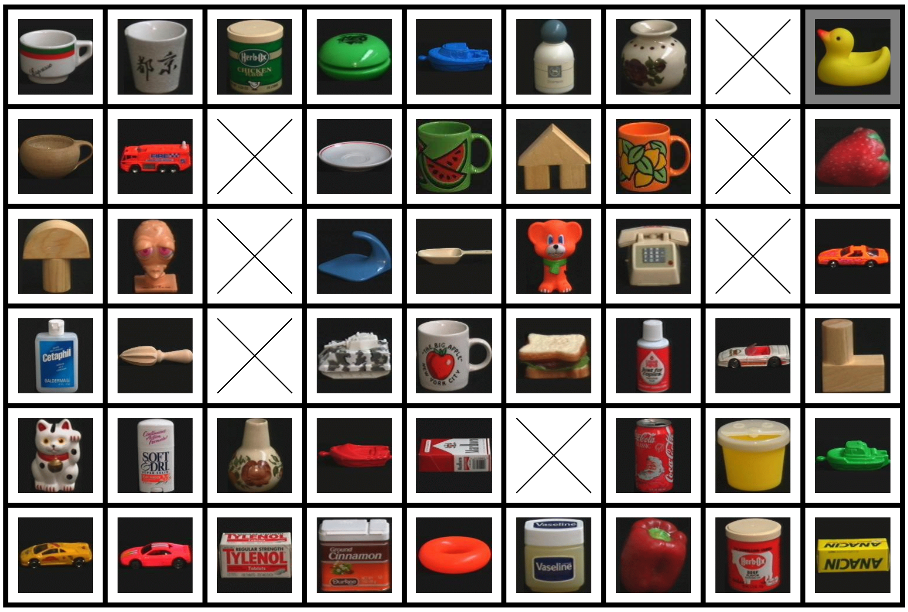


Figure 4.9: Large visual Gridworld topology (with 47 empty cells).

Large visual Gridworld with rotations: This third setup is the same as Large Gridworld, but each time the sensors take a picture of the object, its point of view is randomly chosen in the interval $[0°, 45°]$, which adds complexity to the task. RLVC succeeded after 71 iterations at computing a classifier that distinguishes between 343 perceptual classes by testing 171 different features out of a set of 1141 possible features. This classifier is fine enough to obtain an optimal policy for the task.

These experiments indicate that RLVC can succeed at simple vision-for-action tasks.

4.6.3 Illustration on a Continuous Navigation Task

The experiments above on visual Gridworlds have discrete dynamics. In the current section, we evaluate RLVC on an abstract task that parallels a real-world scenario in which the navigation is continuous. However, we avoid any unnecessary complexity with the visual sensors. As a consequence, the sensor model we use may seem unrealistic; a better visual sensor model will be used in Section 5.1.5.

RLVC has succeeded at solving the continuous, noisy visual navigation task depicted in Figure 4.10. An agent moves inside a maze in which walls are present. The agent is reduced to a single point, so it is always free to move between any two walls. The goal of the agent is to reach as fast as possible one of the two exits of the maze. The set of possible locations is continuous (i.e. subpixel). At each location, the agent has four possible actions: Go up, right, down, or left. Every move is altered by Gaussian noise, the standard deviation of which is 2% the size of the maze. Glass walls are present in the maze. Whenever a move would take the

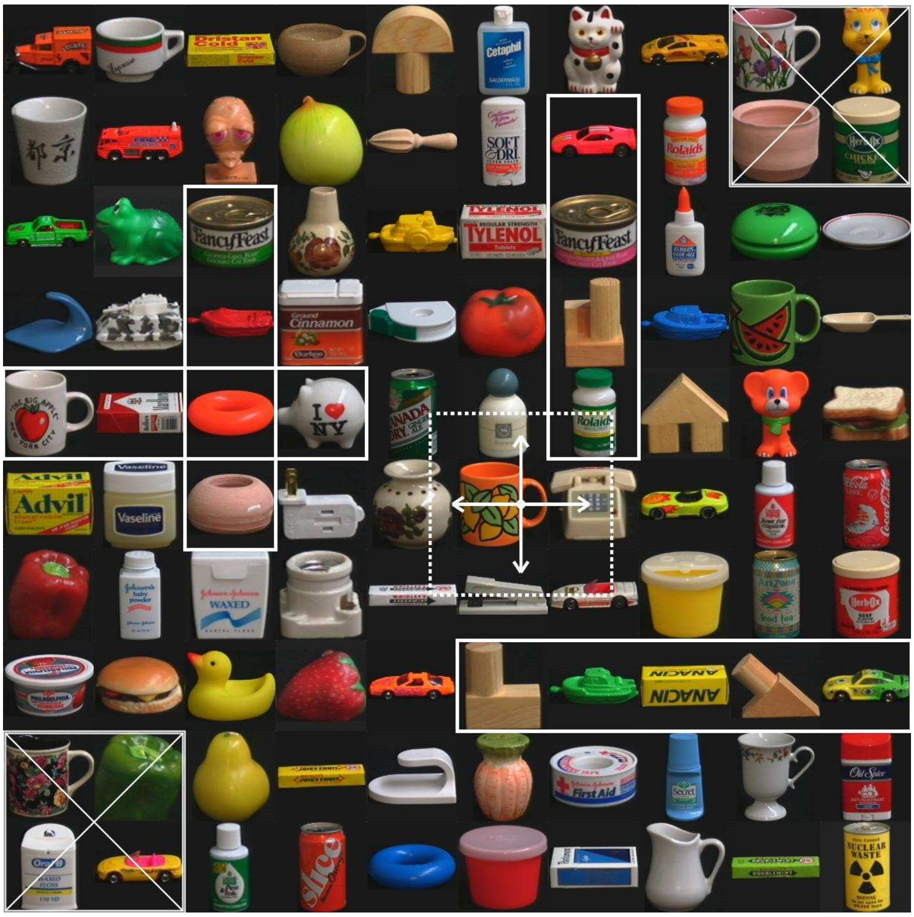


Figure 4.10: A visual, continuous, noisy navigation task. The exits of the maze are indicated by boxes with a cross. Walls of glass are identified by solid lines. The agent is depicted at the center of the figure. Each one of the four possible moves is represented by an arrow, the length of which corresponds to the resulting move. The sensors return a picture that corresponds to the dashed portion of the image.

agent into a wall or outside the maze, its location is not changed. If the position of the agent is initially set inside a wall, it can make no move.

The agent earns a reward of 100 when an exit is reached. Any other move, including the forbidden ones, generates zero reinforcement. When the agent succeeds at escaping the maze, it arrives in a terminal state in which every move gives rise to a zero reinforcement. In this task, γ was set to 0.9. Note that the agent faces the delayed-reward problem, and that it must take the distance to the two exits into consideration for choosing the most attractive one.

The maze has a ground carpeted with a color image of 1280 × 1280 pixels that is an assembly of pictures from the COIL-100 database [NNM96]. The agent does not have direct access to its (x, y) position in the maze. Rather, its sensors take a picture of a surrounding portion of the ground. This portion is larger than the blank areas, which makes the input space fully observable, albeit at limited spatial

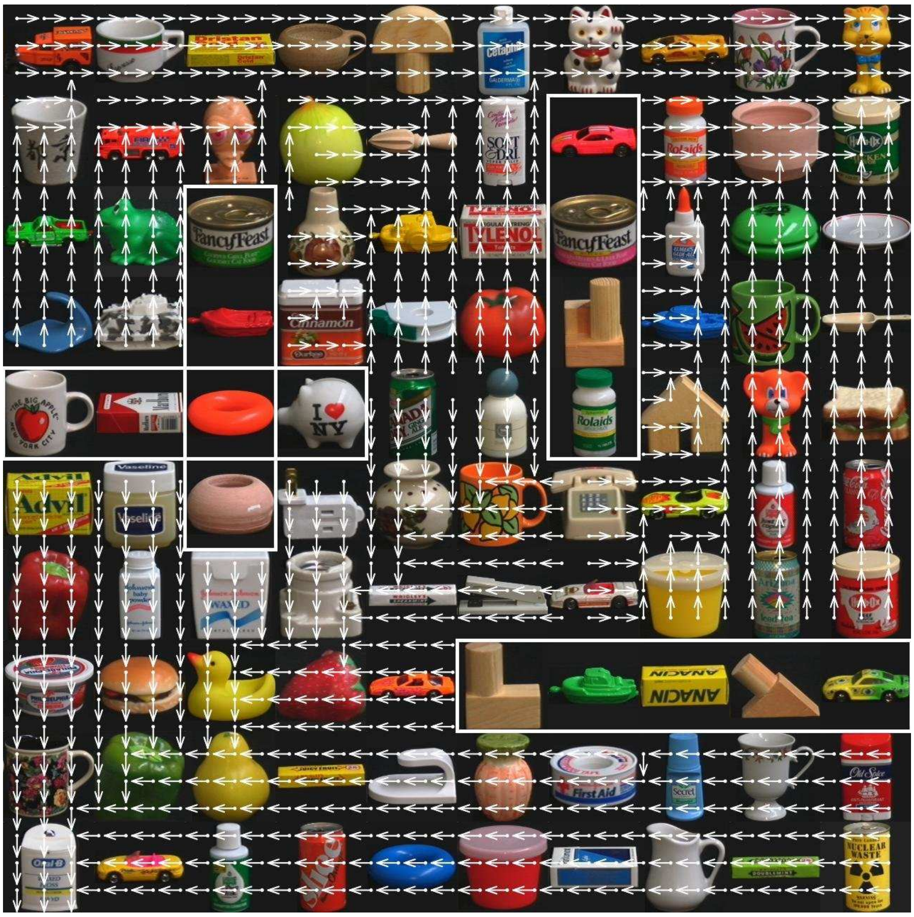


Figure 4.11: The resulting image-to-action mapping $\pi^*(s) = \text{argmax}_{a \in A} Q_k^*(s,a)$, sampled at regularly-spaced points. It manages to choose an optimal action at each location.

resolution. Importantly, the glass walls are transparent, so that the sensors also return the portions of the tapestry that are behind them. Therefore, there is no way for the agent to directly locate the walls. It is obliged to identify them as the regions of the maze in which an action does not change its location.

In this experiment, we have used color differential invariants [GB01] as visual features. The entire tapestry includes 2298 different visual features. RLVC selected 200 features, corresponding to a ratio of 9% of the entire set of possible features. The computation stopped after the generation of 84 percept classifiers (i.e. when k reached 84), which took 35 minutes on a 2.4GHz Pentium IV using a static database of 10,000 interactions. 205 perceptual classes were identified. This is a small number, compared to the number of perceptual classes that would be generated by a discretization of the maze when the agent knows its (x, y) position. For example, a reasonably sized 20×20 grid leads to 400 perceptual classes.

The optimal, deterministic image-to-action mapping that results from the last obtained image classifier $\mathcal{C}_k$ is shown in Figure 4.11. Figure 4.12 compares the optimal value function of the discretized problem with the one obtained through RLVC.

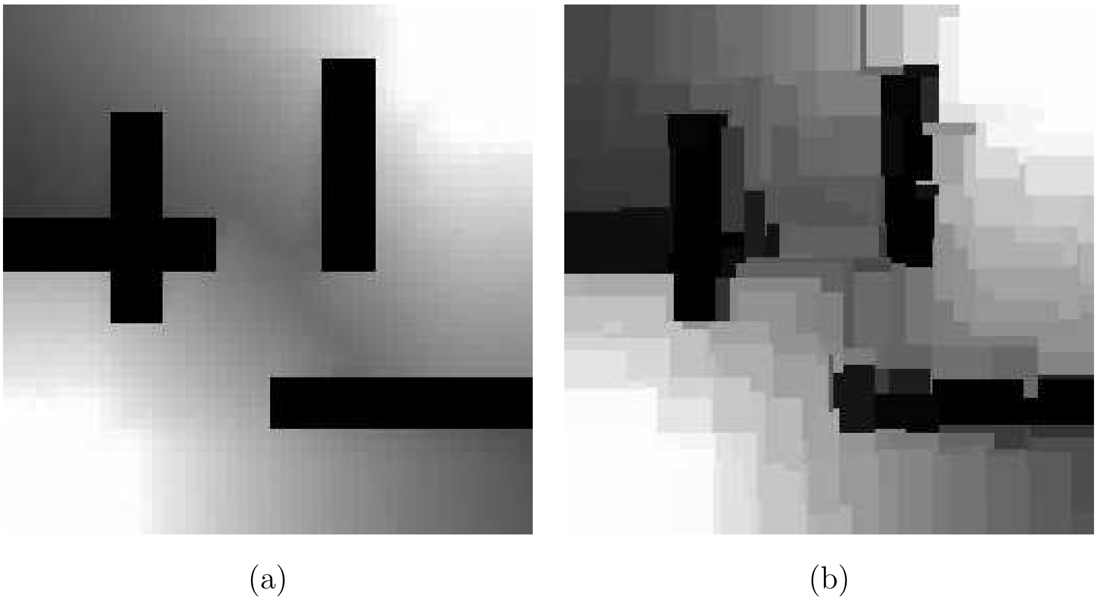


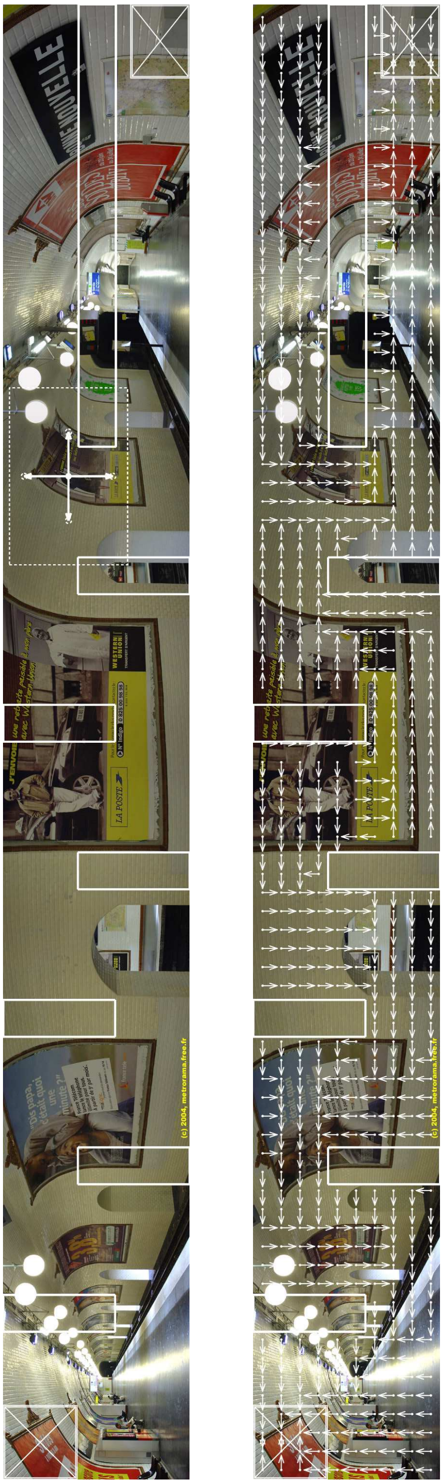
Figure 4.12: (a) The optimal value function, when the agent has direct access to its (x, y) position in the maze and when the set of possible locations is discretized into a 50×50 grid. The brighter the location, the greater its value. (b) The final value function obtained by RLVC.

The similarity between the two pictures indicates the soundness of our approach. Importantly, RLVC operates with neither pretreatment, nor human intervention. The agent is initially not aware of which visual features are important for its task. Moreover, the interest of selecting descriptors is clear in this application: A direct, tabular representation of the Q function considering all the Boolean combinations of features would have $2^{2298} \times 4$ cells.

The behavior of RLVC on real-word images has also been investigated. The navigation rules were kept identical, but the tapestry was replaced by a panoramic photograph of 3041×384 pixels of a subway station, as depicted in Figure 4.13. RLVC took 101 iterations to compute the mapping at the right of Figure 4.13. The computation time was 159 minutes on a 2.4GHz Pentium IV using a static database of 10,000 interactions. 144 distinct visual features were selected among a set of 3739 possible ones, generating a set of 149 perceptual classes. Here again, the resulting classifier is fine enough to obtain a nearly optimal image-to-action mapping for the task.

4.7 Summary

This chapter introduced a novel adaptive-resolution algorithm called *Reinforcement Learning of Visual Classes* (RLVC). RLVC is designed to learn mappings that directly connect complex stimuli to output actions that are optimal for the surrounding environment. The framework of RLVC is general in the sense that it can be applied



(a) (b)

Figure 4.13: (a) A navigation task with a real-world image, using the same conventions as in Figure 4.10. (b) The deterministic image-to-action mapping computed by RLVC.

to any problem that can be formulated as a Markov decision problem. This paradigm can also be motivated from the point of view of purposive vision, and not only as an *ad-hoc* machine learning algorithm.

The learning process behind our algorithms is closed-loop and flexible. The agent takes lessons from its interactions with the environment, similarly to the way living beings learn to solve everyday tasks. RLVC focuses the attention of an embedded reinforcement learning algorithm on highly informative and robust parts of the inputs by testing the presence or absence of perceptual features. The relevant perceptual features are incrementally selected in a sequence of attempts to remove perceptual aliasing: The discretization process targets zero Bellman residuals and is inspired by supervised learning algorithms for building decision trees. Our algorithms are defined independently of any perceptual feature space. The user may choose the used perceptual features as he sees fit. We have shown that the use of appearance-based features (cf. Chapter 3) allows RLVC to learn direct image-to-action mappings.

Our approach unifies adaptive-resolution methods for finite and continuous perceptual spaces. On the one hand, RLVC tests the presence of perceptual features that behave as binary classifiers that discretely divide the perceptual space into two parts. On the other hand, thanks to perceptual feature generators, RLVC samples the perceptual features from a possibly infinite, continuous perceptual feature space. This interesting property is due to the introduction of perceptual feature generators: The only human intervention that is needed by RLVC is the development of a suitable generator. Because of their general nature, perceptual feature generators can be used in a wide variety of domains. For instance, any visual feature generator that was introduced in Chapter 3 can be used to solve any vision-for-action task.

CHAPTER

FIVE

Extensions to RLVC

In this chapter, we build on the basic version of RLVC by proposing two extensions that improve the performance of the algorithm on certain tasks:

1. Fighting overfitting. *Because of its greedy nature, RLVC is highly subject to overfitting. Splitting* one *perceptual class can potentially improve the control policy for* all *the perceptual classes. Therefore, the splitting strategy can get stuck in local minima in the original description of RLVC: Once a split is made that subsequently proves useless, it cannot be undone. Thus, the first extension that is discussed in the current chapter consists in integrating a forgetting mechanism with RLVC. Experiments indeed show an improvement in the generalization abilities, as well as a reduction of the number of perceptual classes and selected features [JP05d].*

2. Generating more distinctive visual features. *When RLVC is applied to vision-for-action tasks, its efficacy clearly depends on the discriminative power of the visual features. If their power is insufficient, the algorithm will not be able to completely remove the aliasing, and will thus produce sub-optimal control policies. Practical experiments on simulated visual navigation tasks exhibit this deficiency, as soon as the number of detected visual features is reduced or as features are made more similar by using a less sensitive metric. The second extension to RLVC that is considered consists in constructing highly informative* spatial combinations *of visual features on demand, when individual local features alone are insufficient [JSP05].*

5.1 Compacting the Percept Classifiers

We have previously written that limiting the number of refinements per round of RLVC is generally useful. The first extension to RLVC pushes this reasoning further, and consists in providing RLVC with the possibility of aggregating perceptual classes that share similar properties. This leads to a scheme that combines refining steps (local optimization) with aggregation phases (prevention of overfitting).

Doing so has at least three potential benefits: (a) Useless features are discarded, which enhances generalization capabilities; (b) RLVC can reset the search for good features so as to avoid local optima; and (c) the number of samples available to the embedded RL algorithm for each perceptual class is increased, resulting in better percept-to-action mappings.

5.1.1 Equivalence Relations in Markov Decision Processes

Since we apply an embedded RL algorithm at each stage k of RLVC, properties such as the optimal value function $\hat{V}_k^*(c)$, the optimal state-action value function $\hat{Q}_k^*(c,a)$ and the optimal percept-to-action mapping $\hat{\pi}_k^*(c)$ are known for each mapped MDP $\mathcal{M}_k$, where $c \in S_k$ and $a \in A$. Using these properties, it is easy to define a whole range of equivalence relations between the perceptual classes. For instance, given a threshold $\varepsilon \in \mathbb{R}^+$, we list below three possible equivalence relations for a pair of perceptual classes $(c, c') \in S_k \times S_k$:

Optimal Value Equivalence:

$$|\hat{V}_k^*(c) - \hat{V}_k^*(c')| \leq \varepsilon.$$

Optimal Policy Equivalence:

$$|\hat{V}_k^*(c) - \hat{Q}_k^*(c', \hat{\pi}_k^*(c))| \leq \varepsilon \text{ and } |\hat{V}_k^*(c') - \hat{Q}_k^*(c, \hat{\pi}_k^*(c'))| \leq \varepsilon.$$

Optimal State-Action Value Equivalence:

$$(\forall a \in A)\ |\hat{Q}_k^*(c,a) - \hat{Q}_k^*(c',a)| \leq \varepsilon.$$

We therefore propose to modify RLVC so that, periodically, perceptual classes that are equivalent with respect to one of these criteria are merged together. This way, RLVC alternatively splits and merges perceptual classes. The compaction phase should not be done too often, in order to allow exploration. To the best of our knowledge, this possibility has not been investigated yet in the framework of adaptive-resolution methods in reinforcement learning.

5.1.2 Decision Trees are not Expressive Enough

In the original version of RLVC, the perceptual classes correspond to the leaves of a decision tree. When using decision trees, the aggregation of perceptual classes can only be achieved by starting from the bottom of the tree and recursively collapsing leaves, until dissimilar leaves are found. This operation is very close to *post-pruning* in the framework of decision trees for machine learning [BFS84]. In practice, this means that classes that have similar properties, but that can only be reached from one another by making a few number of hops upwards then downwards, are extremely unlikely to be matched. This greatly reduces the interest of exploiting the equivalence relations.

This drawback is due to the rather limited expressiveness of decision trees. In a decision tree, each perceptual class corresponds to a conjunction of perceptual feature literals, which defines a path from the root of the decision tree to one leaf. To take full advantage of the equivalence relations, it is necessary to associate, to each perceptual class, an arbitrary *union* of conjunctions of perceptual features. Indeed, when exploiting the equivalence relations, the perceptual classes are the result of a sequence of conjunctions (splitting) and disjunctions (aggregation). Thus, a more expressive data structure that would be able to represent general, arbitrary Boolean combinations of perceptual features is required. Such a data structure is introduced in the next section.

5.1.3 An Excursion into Computer-Aided Verification

Thanks to significant advances in digital-processor technology in the past few decades, embedded controllers are nowadays widely integrated at the automation level of factories and in common-life electronic appliances. Thus, there is a growing interest in the automatic verification of such devices, which has led to the development of the field of *computer-aided verification*. One of the basic techniques of computer-aided verification is *reachability analysis* that consists in computing the set of all states that can be reached by a program from a fixed set of initial conditions. When this set is computed, it is possible to test whether it intersects a set of undesirable states that cause a failure of the system. This leads to a *safety analysis* of programs.

As a consequence, there is a fundamental need in computer-aided verification for representing a set of states of a given program. A whole range of methods for representing the state space of richer and richer domains have been developed over the last few years, such as *Finite Unions of Polyedra* [Sch86], *Algebraic Decision Diagrams* [BFG$^+$93b], *Number* and *Queue Decision Diagrams* [Boi99], *Upward Closed Sets* [DR00] and *Real Vector Automata* [BBR97, BRW98, BJW05][1].

Representation systems for general Boolean functions (i.e. mappings from $\mathcal{B}^n$ to $\mathcal{B}$) have been extensively studied, since Boolean functions can abstract the behavior of logical, discrete electronic devices. Unfortunately, canonical representations like truth tables or Karnaugh maps are quite impractical because they need 2^n memory cells for their representation of a function with n Boolean arguments. This has motivated the development of the highly successful *(Ordered) Binary Decision Diagram* (BDD) representation [Bry86, Bry92]. BDDs are a compact, efficient and widespread way to represent and manipulate arbitrary Boolean functions. The well-known computer-aided verification tool SPIN notably embeds an implementation of elaborate BDD structures [SPI06].

A BDD is a directed, acyclic graph, the internal nodes of which are labeled by a test on one Boolean variable. Each internal node has two children, one that cor-

[1] Note that one of our personal contributions that is outside the scope of this dissertation has consisted in showing, through topological arguments, how the difficult and delicate-to-implement algorithms for handling Real Vector Automata can be simplified by using a restricted class of automata on infinite words [Jod01, BJW01, BJW05]. This has allowed us to achieve reachability analysis of linear hybrid systems [Jod02], that has later been extended with acceleration techniques so as to compute state spaces that can only explored through an infinite sequence of discrete transformations [BJH03].

p	q	r	s	$f(p,q,r,s)$
F	F	F	F	F
F	F	F	T	F
F	F	T	F	F
F	F	T	T	T
F	T	F	F	F
F	T	F	T	F
F	T	T	F	F
F	T	T	T	T
T	F	F	F	F
T	F	F	T	F
T	F	T	F	F
T	F	T	T	T
T	T	F	F	T
T	T	F	T	T
T	T	T	F	T
T	T	T	T	T

Figure 5.1: A truth table that defines a mapping $f : \mathcal{B}^4 \mapsto \mathcal{B}$.

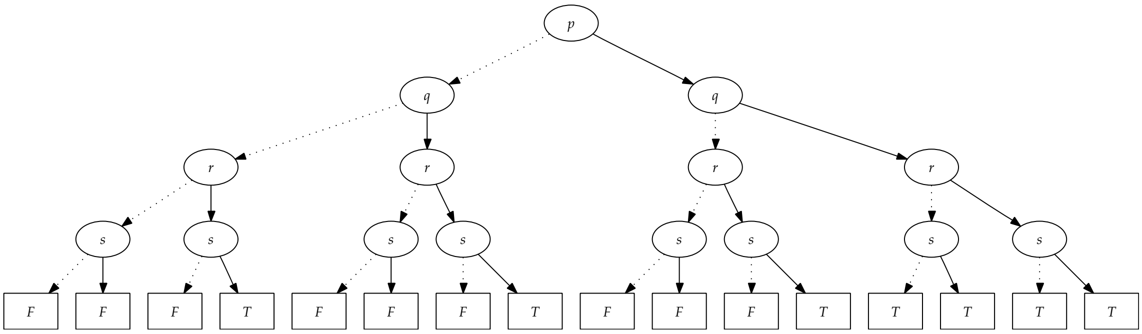


Figure 5.2: The *Shannon decomposition* [Sha49] that corresponds to the truth table of Figure 5.1. In this picture, a dotted line indicates the assignment of the Boolean value **false**, whereas a solid line corresponds to an assignment to **true**.

responds to an assignment of **true** to the corresponding decision variable, and the other that corresponds to an assignment of **false**. The terminal nodes of the graph are labeled either by **true** or by **false**, which defines the Boolean output of the function. Thus, each path from the root to a leaf corresponds to an evaluation of the Boolean function for a specific assignment of its input variables. BDDs make the assumption that all the Boolean variables appear in the same order on all paths from the root[2]. This makes possible the normalization to a unique, canonical form. For example, consider a Boolean function, the truth table of which is given in Figure 5.1. The normalization process for this function is illustrated by Figures 5.2 and 5.3. As a consequence, the testing of functional properties such as satisfiability and equivalence become straightforward: The equivalence of two Boolean functions can be simply tested by acyclic graph matching, and testing for satisfiability reduces to a comparison with the constant function **false**. Furthermore, useful Boolean operations such as conjunctions, projections, disjunctions or evaluations can be directly achieved on this symbolic representation.

[2] Strictly speaking, this assumption only holds for *Ordered* BDDs that have been introduced by Bryant [Bry86, Bry92]. However, Ordered BDDs are by far the most useful category of BDDs because of their normal form that is induced by the ordering assumption.

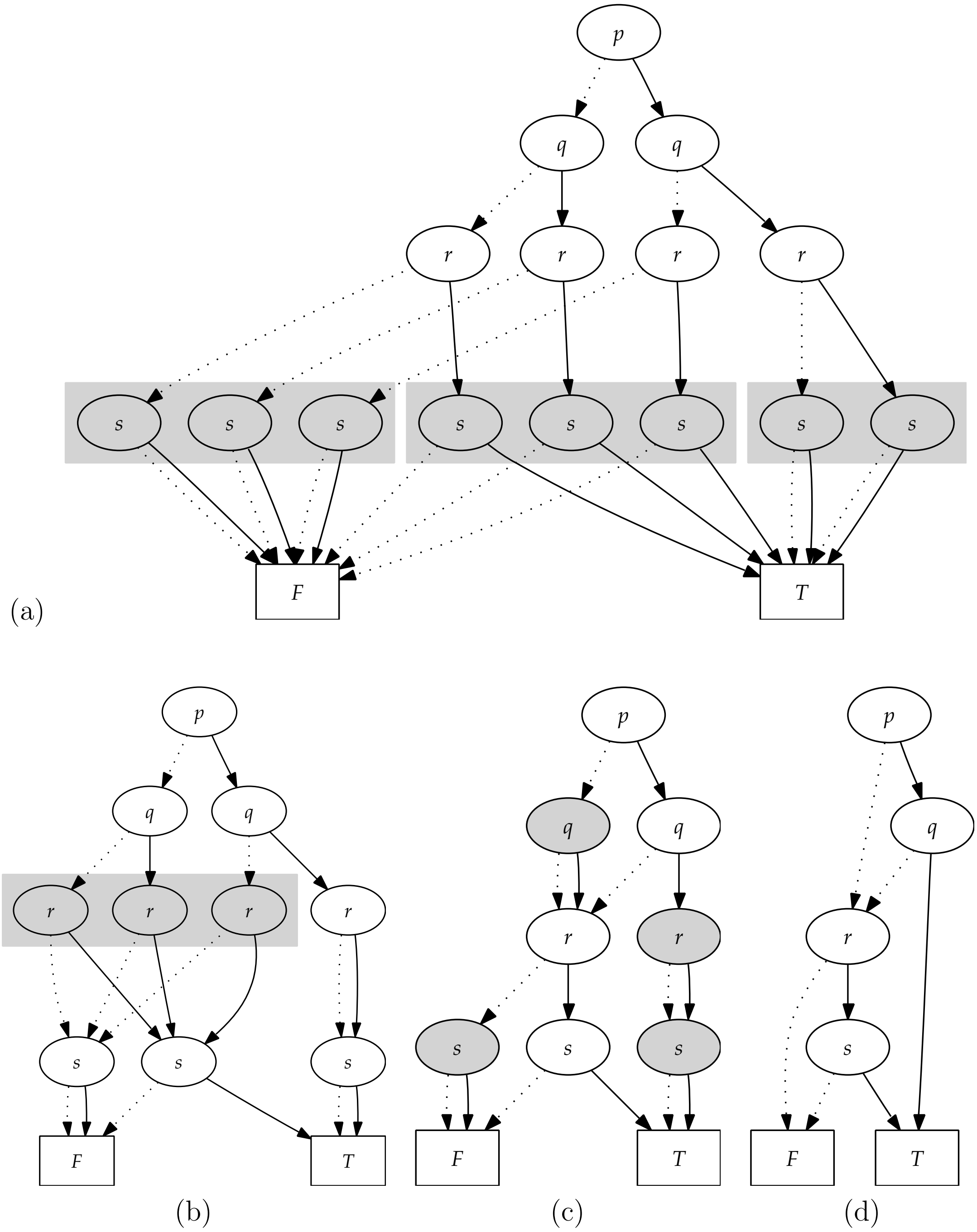


Figure 5.3: Normalization of the Shannon decomposition of Figure 5.2. (a) First, the tree structure is turned into an acyclic graph by merging the terminal nodes, so that only two terminal nodes are kept (one for a **true** output, the other for **false**). (b,c) Secondly, identical subtrees are recursively merged and shared, starting with the last variable in the ordering. (d) Finally, useless tests that lead to the same node are discarded. The last graph is the resulting Binary Decision Diagram. Obviously, all the graphs in this sequence represent the same Boolean function.

But in order to benefit from these advantages, we have to define an ordering on the variables for all functions to be represented. The size of a BDD representation can be highly dependent on this ordering. Unfortunately, the problem of computing an optimal ordering is co-NP-complete [Bry86].[3] Nevertheless, automatic heuristics can in practice find orderings that are close to optimal, and are implemented in all the software libraries for manipulating BDDs. This is interesting in our case, since reducing the size of the BDD potentially discards irrelevant variables, which corresponds to removing useless perceptual features.

As a consequence, BDDs are a particularly well-suited tool for our framework. In the sequel, we will write $\mathcal{B}(n)$ to designate the set of BDDs with n Boolean inputs, and

$$B\Big|_{\boldsymbol{b}} \tag{5.1}$$

will represent the application of some BDD B to the vector of Booleans $\boldsymbol{b} \in \mathcal{B}^n$.

5.1.4 Embedding BDDs inside RLVC

We propose to replace the single decision tree that represents the percept classifier and that is progressively refined, by a set of Binary Decision Diagrams. More precisely, one BDD is assigned to each perceptual class:

Definition 5.1. Let

$$\hat{F}_k = \left(f_1^{(k)}, \ldots, f_{n_k}^{(k)}\right) \tag{5.2}$$

be a finite, ordered subset of n_k perceptual features from F_S. Let also

$$\hat{\mathcal{C}}_k : C_k \mapsto \mathcal{B}(n_k) \tag{5.3}$$

be a mapping that associates one BDD with each perceptual class, the set of all these BDDs defining a partition[4] of $\mathcal{B}^{n_k}$. Then, a *BDD-based percept classifier* is a percept classifier $\mathcal{C}_k : S \mapsto S_k$ that is defined as:

$$\mathcal{C}_k(s) = c \text{ if and only if } \hat{\mathcal{C}}_k(c)\Big|_{\boldsymbol{b}(s)} = \textbf{true}, \tag{5.4}$$

for each percept $s \in S$, where $\boldsymbol{b}(s) \in \mathcal{B}^{n_k}$ is the vector of Booleans that corresponds to the outputs of the perceptual feature detector for all the features in $\hat{F}_k$:

$$\boldsymbol{b}(s) = \begin{pmatrix} \mathcal{D}_S\left(s, f_1^{(k)}\right) \\ \vdots \\ \mathcal{D}_S\left(s, f_{n_k}^{(k)}\right) \end{pmatrix}. \tag{5.5}$$

The embedding of BDD-based percept classifiers inside the RLVC process requires two modifications of Algorithm 4.1: class refinement and class aggregation.

[3] A problem is a member of co-NP if and only if its complement problem is in complexity class NP.

[4] This set is a partition of $\mathcal{B}^{n_k}$ if for any $\boldsymbol{b} \in \mathcal{B}^{n_k}$, there exists one and only one perceptual class c such that $\hat{\mathcal{C}}_k(c)$ assigns **true** to $\boldsymbol{b}$.

Algorithm 5.1 — Refining a BDD-based percept classifier

```
bdd-refining($\mathcal{C}_k, c, f^*$) :–
  $i \leftarrow \bot$
  for $j \leftarrow 1$ to $n_k$ do
    if $f_j^{(k)} = f^*$ then
      $i \leftarrow j$
    end if
  end for
  if $i = \bot$ then
    {$f^*$ is an new feature}
    for $j \leftarrow 1$ to $m_k$ do
      Add a new decision variable to $\hat{\mathcal{C}}_k[j]$
    end for
    $n_k \leftarrow n_k + 1$
    $i \leftarrow n_k$
    $\hat{F}_k \leftarrow \left(\hat{F}_k \;;\; f^*\right)$
  end if
  $m_k \leftarrow m_k + 1$
  $\hat{\mathcal{C}}[m_k] \leftarrow \hat{\mathcal{C}}_k[c] \wedge i$
  $\hat{\mathcal{C}}[c] \leftarrow \hat{\mathcal{C}}_k[c] \wedge \neg i$
```

Class refinement: The operation of refining a perceptual class c with a perceptual feature f^* consists in three steps:

1. The perceptual feature f^* is matched[5] against all the perceptual features inside the set $\hat{F}_k$. This step allows RLVC to realize that a perceptual feature has been previously used. It might reduce the size of the BDDs by avoiding redundant tests, while reducing the number of variables upon which the BDDs are defined.
2. If the feature f^* is previously unseen, it is added to $\hat{F}_k$. Furthermore, an additional input variable is added to all the BDDs that are stored inside $\hat{\mathcal{C}}_k$: This is a Cartesian product operation, which is easily achievable on BDDs.
3. The old perceptual class c is removed from the percept classifier, and is replaced by two new perceptual classes $c_\oplus$ and $c_\ominus$. The BDDs that are associated with $c_\oplus$ and $c_\ominus$ in $\hat{\mathcal{C}}_k(c)$ are defined as follows:

$$\hat{\mathcal{C}}_k(c_\oplus) = \hat{\mathcal{C}}_k(c) \wedge f^* \tag{5.6}$$

$$\hat{\mathcal{C}}_k(c_\ominus) = \hat{\mathcal{C}}_k(c) \wedge \neg f^*. \tag{5.7}$$

The resulting operations are outlined in Algorithm 5.1.

[5] This requires the introduction of an equivalence relation that is defined over the perceptual feature space. In the case of visual features, this could be a simple threshold on the Euclidean distance.

Algorithm 5.2 — Compacting a BDD-based percept classifier

```
post-process $(\mathcal{C}_k)$ :-
  {Detect equivalent perceptual classes}
  $R \leftarrow \{\}$    {Equivalence classes}
  $S \leftarrow S_k$    {Perceptual classes not considered so far}
  while $S \neq \emptyset$ do
    $c \leftarrow$ Select a perceptual class in $S$
    $S, T \leftarrow S \setminus \{c\}, \{c\}$
    for $c' \in S$ do
      if equivalent-classes$(c, c')$ then
        $S, T \leftarrow S \setminus \{c'\}, T \cup \{c'\}$
      end if
    end for
    $R \leftarrow R \cup \{T\}$
  end while
  {Merge equivalent classes into a new BDD-based percept classifier}
  $m'_k = |R|$
  $i \leftarrow 1$
  while $R \neq \emptyset$ do
    $T \leftarrow$ Select an equivalence class in $R$
    $R \leftarrow R \setminus \{T\}$
    $i, c' \leftarrow i + 1, c_i'^{(k)}$
    $\hat{\mathcal{C}}'_k(c') \leftarrow$ true
    for $c \in T$ do
      $\hat{\mathcal{C}}'_k(c') \leftarrow \hat{\mathcal{C}}'_k(c') \vee \hat{\mathcal{C}}_k(c)$
    end for
  end while
  Replace the BDD-based percept classifier $\langle \hat{F}_k, m_k, \hat{\mathcal{C}}_k \rangle$ by $\langle \hat{F}_k, m'_k, \hat{\mathcal{C}}'_k \rangle$
  {Remove irrelevant perceptual features and decision variables}
  for $i \leftarrow 1$ to $n_k$ do
    $b \leftarrow$ false
    for $c \in S_k$ do
      if $\hat{\mathcal{C}}_k(c)$ makes at least one test on the $i$th variable then
        $b \leftarrow$ true
      end if
    end for
    if not $b$ then
      Remove $f_i^{(k)}$ from $\hat{F}_k$
      $n_k \leftarrow n_k - 1$
      for $c \in S_k$ do
        Remove the $i$th variable from $\hat{\mathcal{C}}_k(c)$ through a projection
      end for
    end if
  end for
```

Class aggregation: The aggregation of similar perceptual classes takes place in the `post-process`($\mathcal{C}_k$) component that was left empty in the basic version of RLVC. When two identical perceptual classes c_1 and c_2 are detected (with respect to a fixed equivalence relation), they are merged into a single class c, the BDD of which is defined as:

$$\hat{\mathcal{C}}_k(c) = \hat{\mathcal{C}}_k(c_1) \vee \hat{\mathcal{C}}_k(c_2). \tag{5.8}$$

Every time a merging operation takes place, it is advised to perform a variable reordering so as to minimize the memory requirements.

A possible, interesting side-effect of merging perceptual classes is that this operation might leave some variable inside $\hat{F}_k$ unused. Therefore, each variable in the set $\hat{F}_k$ is tested so as to find a BDD that uses this variable. If the variable is unused, then it is deleted from the set $\hat{F}_k$ and is taken off all the BDDs through a projection operation. As a result, irrelevant perceptual features (i.e. features that are found to be not discriminant) are progressively eliminated. The corresponding piece of pseudo-code is given in Algorithm 5.2.

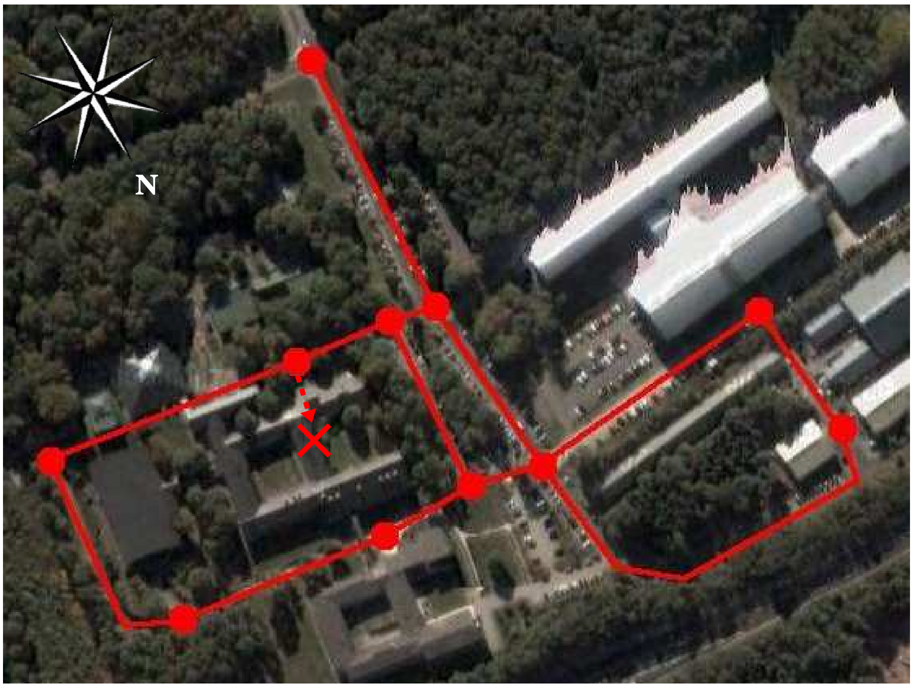
This concludes the discussion of our techniques for avoiding overfitting inside RLVC. One must however be aware that, as already pointed out by McCallum, the distinctions that are necessary to *learn* an optimal percept-to-action mapping are generally finer than the distinctions that are necessary to *represent* the resulting optimal mapping [McC96]. In practice, to allow exploration, this means that aggregation phases should not occur too often. Therefore, this is a heuristic technique that might prove to be useful, but that is to use with caution because it may prevent the convergence of RLVC.

5.1.5 Navigation around Montefiore Institute

We have applied the modified version of RLVC to another simulated navigation task. In this task, the agent moves between 11 spots of the campus of the University of Liège (cf. Figure 5.4). Every time the agent is at one of the 11 locations, its body can aim at four possible orientations: North, South, West, East. The physical state space is therefore of size $11 \times 4 = 44$. The agent has three possible actions at its disposal: Turn left, turn right, go forward to the next intersection. Its goal is to enter a specific building, where it will obtain a reward of $+100$. Turning left or right induces a penalty of -5, and moving forward, a penalty of -10. The discount factor γ is set to 0.8. The optimal control policy is not unique: One of them is depicted in Figure 5.5.

The agent does not have direct access to its position and orientation. Rather, it only perceives a picture of the area that is in front of it (cf. Figure 5.6). Thus, the agent has to connect an input image to the appropriate action without explicitly knowing its geographical localization. The full database of images that has been collected is available on the Internet [Jod05].

We have used SIFT keypoints as visual features [Low04]. Before applying the SIFT keypoint detector, we have downscaled each of the 1,056 color images to a



(c) Google Map

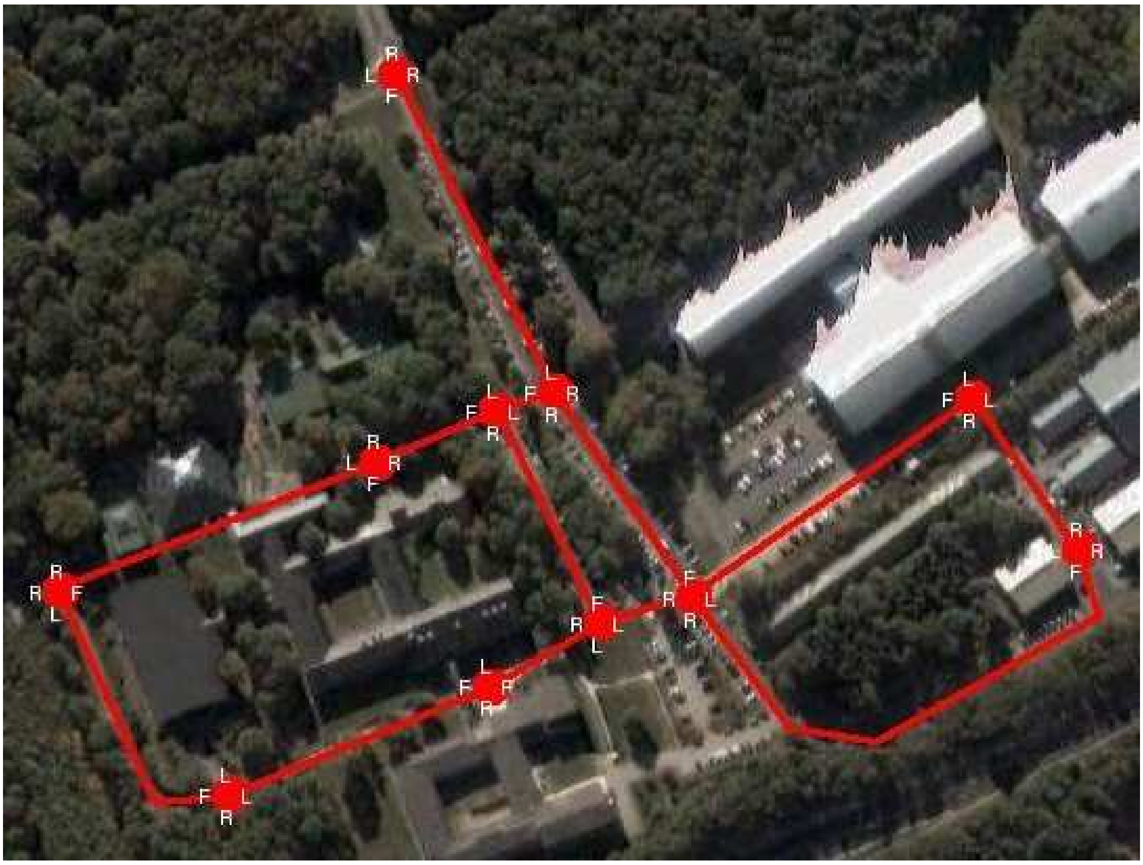
Figure 5.4: The Montefiore campus in Liège. Red spots correspond to the places between which the agent moves. The agent can only follow the links between the different spots. Its goal is to enter the Montefiore Institute, that is labeled by a red cross, where it obtains a reward of +100.

gray-level image of size 160 × 120. Thresholding on the Mahalanobis distance gave rise to a set of 13,367 distinct visual features. Both versions of RLVC have been applied on a static database of 10,000 interactions that has been collected using a fully randomized exploration policy. The same database of interactions is used throughout the entire algorithms, and this database only contains images that belong to the learning set.

The original version of RLVC has identified 281 perceptual classes by selecting 264 SIFT features. The error rate on the computed visual policy (i.e. the proportion of sub-optimal decisions when the agent is presented all the possible stimuli in the image databases) was 0.1% on the learning set and 8% when the images of the test set are used[6]. Note that this difference between learning and test sets clearly indicates the presence of overfitting effects. The detail of the corresponding statistics are reported in Figure 5.7.

The modified version of RLVC was then applied, with one compacting stage every 10 iterations of RLVC. It was experimentally observed on this task that using the conjunction of the first two criteria of Section 5.1.1 (optimal value equivalence and

[6] This is with respect to the optimal policy when the agent has direct access to its position and viewing direction.

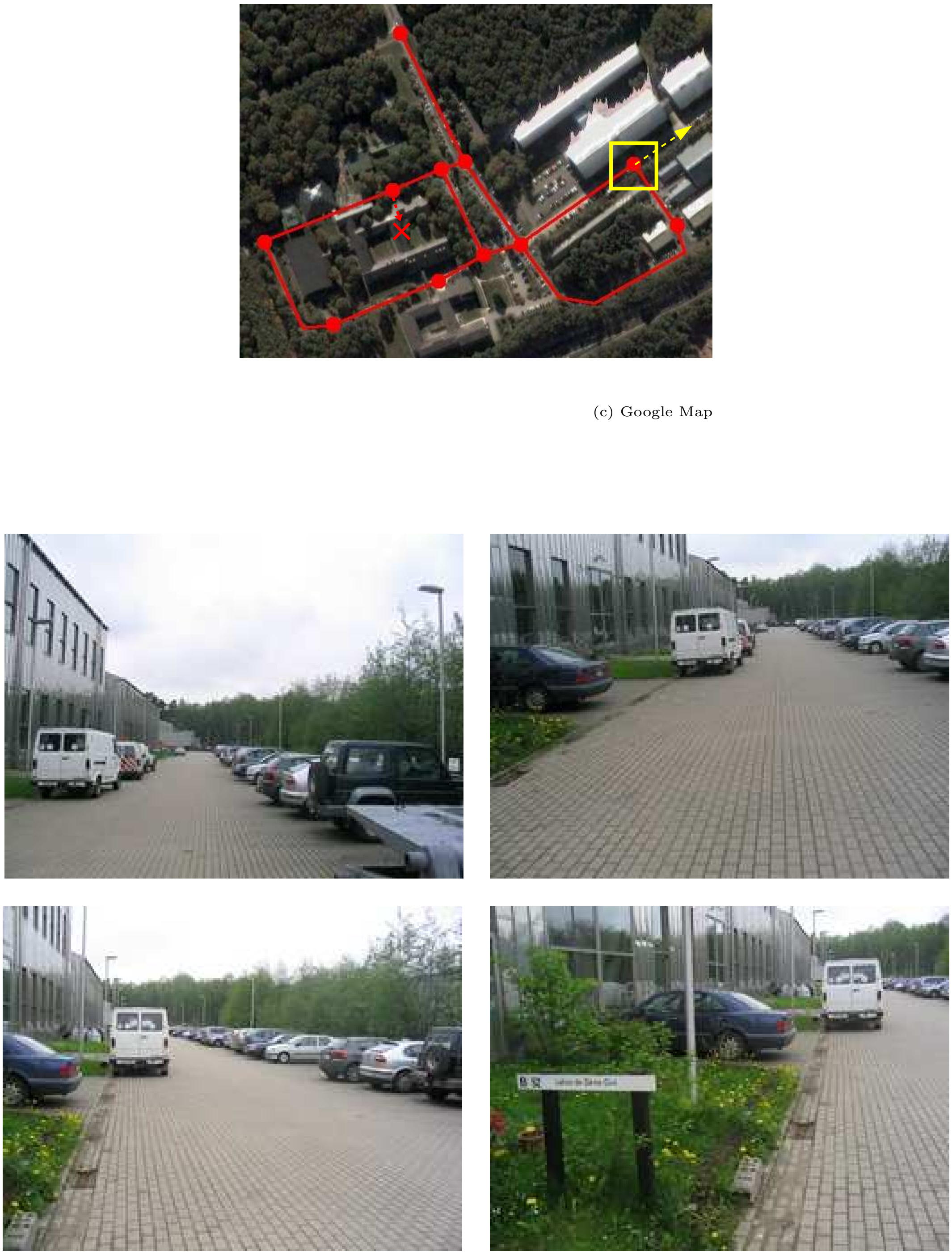


(c) Google Map

Figure 5.5: One of the optimal, deterministic control policies for the Montefiore navigation task. For each state, we have indicated the optimal action (the letter “F” stands for “move forward”, “R” for “turn right” and “L” for “turn left”). This policy has been obtained by applying a standard RL algorithm to the scenario in which the agent has direct access to its position and viewing direction.

optimal policy equivalence) as an equivalence relation tends to lead to interesting performance. Intuitively, states are thus considered dissimilar if they differ either in their utility or in their optimal action.

The results are shown in Figure 5.8 and are clearly superior. There is no error on the learning set anymore, while the error rate on the test set is 4.5%. The number of selected features is reduced to 171. Furthermore, the resulting number of perceptual classes becomes 59, instead of 281. Thus, there is a large improvement in the generalization abilities, as well as a reduction of the number of perceptual classes and selected visual features. Interestingly, the number of perceptual classes (59) is very close to the number of physical states (44), which indicates that the algorithm starts to learn a physical interpretation for its percepts.



(c) Google Map

Figure 5.6: The percepts of the agent. For each possible location and each possible viewing direction, a database of 24 images of size 1024 × 768 with significant viewpoint changes has been collected [Jod05]. Those 44 databases have been randomly divided into a learning set of 18 images and a test set of 6 images. Four different percepts are shown, that correspond to the location and viewing direction marked by a yellow square on the top image.

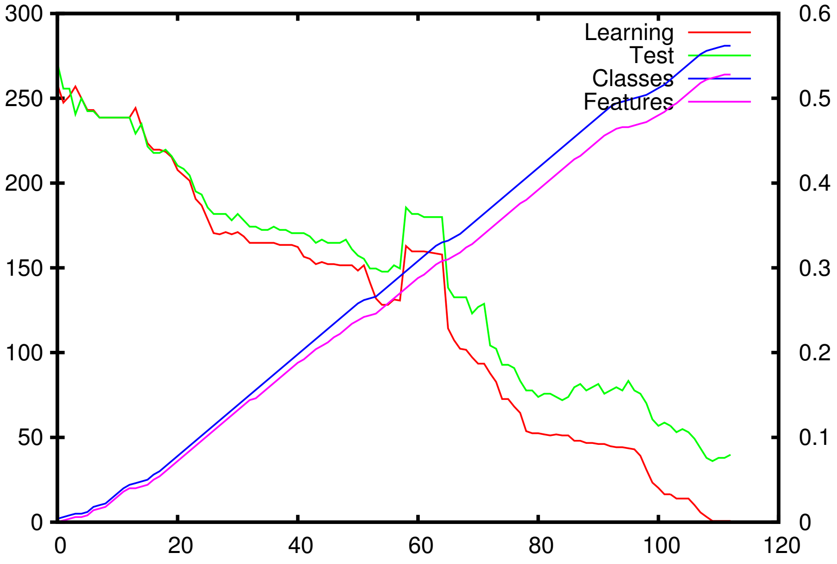


Figure 5.7: Statistics of RLVC as a function of the step counter k. The red (resp. green) plot corresponds to the error of the computed policy on the learning (resp. test) images — the corresponding axis is on the right of the plot. The number of perceptual classes (resp. selected features) is plotted in blue (resp. purple) — the corresponding axis is on the left of the plot.

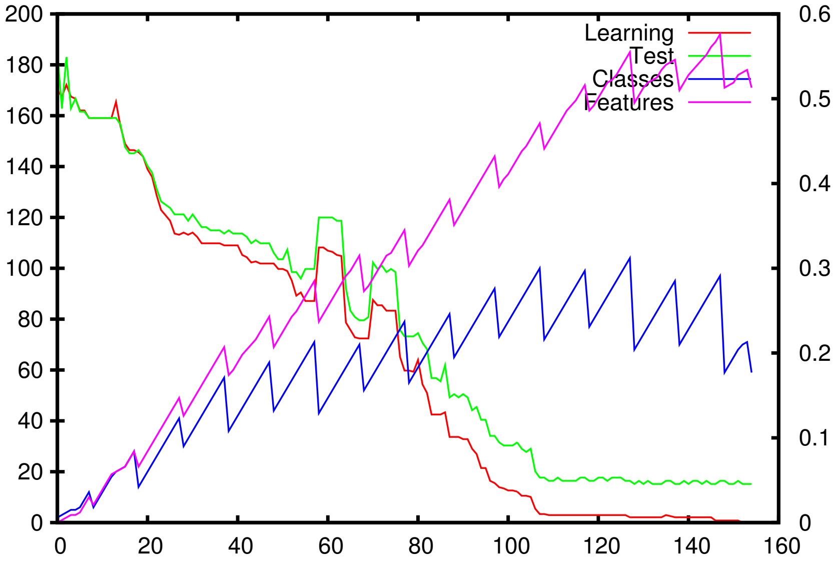


Figure 5.8: Statistics of the extended version of RLVC as a function of the step counter k, using the same conventions as Figure 5.7. The decreases in the blue and purple curves indicate the compacting phases. It is interesting to highlight the fact that a compacting phase tends to reduce the errors. These results are superior to the scores of the original RLVC.

5.1.6 Discussion

To summarize, experimental results show an improvement when visual policies are regularly compacted. Thus, compacting visual policies is probably an important step to deal with realistic visual tasks, if an iterative splitting process is applied. The price to pay is of course a higher computational cost. Future work will focus on a theoretical justification of the used equivalence relations. This implies bridging the gap to the theory of MDP minimization [GDG03].

Finally, we also note that an implementation of percept classifier through Algebraic Decision Diagrams instead of BDDs may ease the implementation, and reduce the computational and memory requirements. Algebraic Decision Diagrams (ADDs) [BFG$^+$93b] are a generalization of BDDs to multi-variate outputs: They can directly represent mappings from Boolean inputs $\mathcal{B}^n$ to a finite set of classes. Therefore, an ADD-based percept classifier would embed only a single ADD instead of one BDD per perceptual class. However, ADD-based percept classifiers would lead exactly to the same results as BDD-based percept classifiers, because the performance of the resulting percept-to-action mapping only depends on the used equivalence relation.

5.2 Learning Hierarchies of Visual Features

After Chapter 4 and Section 5.1 that have considered general perceptual feature spaces, we now introduce an extension to RLVC that is exclusively related to vision-for-action tasks. Most objects encountered in the world are composed of a number of distinct constituent parts (e.g. a face contains a nose and two eyes, a phone possesses a keypad). These parts are themselves recursively composed of other sub-parts (e.g. an eye contains an iris and eyelashes, a keypad is composed of buttons). Such a hierarchical physical structure certainly imposes strong constraints on the spatial disposition of the visual features. This idea was already illustrated in Figure 3.5 (page 55).

Following this line of reasoning, we propose a second extension to RLVC that consists in iteratively constructing highly informative *spatial combinations* of visual features. This research is promising for it permits the construction of features at increasingly higher levels of discriminative power, enabling us to tackle visual tasks that are unsolvable using individual point features alone. To the best of our knowledge, this extension to RLVC appears to be one of the first attempts to build visual feature hierarchies in a closed-loop, interactive and task-driven learning process. The most closely related work is Piater's doctoral dissertation [Pia01].

Also note that the interest of this research can also be motivated from a neuropsychological point of view. Wallis and Bülthoff indeed write that neuropsychological studies on this topic suggest two major hypotheses:

> “The first is that objects are represented in a mosaic-like form in which objects are encoded by combinations of complex, reusable features. rather than two-dimensional templates, or three-dimensional mod-

> els. The second hypothesis is that transform-invariant representations of objects are learned through experience, and that this learning is affected by the temporal sequence in which different views of the objects are seen, as well as by their physical appearance." [WB99]

5.2.1 Related Work

The idea of combining visual features that are spatially close to each other goes back to *semi-local constraints* [SM97]. Semi-local constraints are a pragmatic approach to reduce the number of false positives when assessing the presence of visual features. They consist in matching two visual features only if they share a sufficient number of common visual features in their close neighborhood. These methods are reviewed in Forsyth and Ponce's textbook [FP03, Section 23.1.4]. Evolutions of these approaches has been used in many areas of computer vision. We notably point out the use of generalized semi-local constraints for the tracking of a geometric model of a soccer player [GHPV05].

This idea has later been generalized to *hierarchical models* of spatial arrangements of features. Semi-local constraints can be understood as hierarchical models with only one level. Since the seminal work by Forsyth et al. about object detection [FHI99], such hierarchical models have become increasingly popular [WK02, BT05, EU05]. For instance, Scalzo and Piater propose to build a probabilistic, graph-based hierarchy of visual features [SP05, SP06]. They detect the presence of such a model through Nonparametric Belief Propagation [SIFW03]. Other graphical models have been proposed for representing articulated structures, such as pictorial structures [FH05, KTZ04]. Similarly, the constellation model represents objects by parts, each modeled in terms of both shape and appearance by Gaussian probability density functions [PFZ03].

The reason for the interest in hierarchical models comes from the fact that visual features, as detected by interest point detectors and as characterized by local description generators, are not able to describe the *shape* of the observed objects. As a consequence, there is a recent, general agreement that higher-level appearance-based representations are highly desirable, notably to deal with the problem of image categorization. Our work contrasts with previous approaches in the sense that the generation of so-called *composite features* is driven by the task to be solved. In RLVC, we have additional information, namely the reinforcement signal, that should be used to keep only visual components that are meaningful for the task.

5.2.2 An Unbounded Hierarchy of Spatial Relationships

From an algorithmic point of view, we introduce, in the current section, a more elaborate perceptual feature generator $\mathcal{G}_S$ that can generate a hierarchy of visual features simultaneously with the percept classifier. As soon as sufficiently informative visual feature cannot be extracted anymore, the algorithm tries to combine two visual features in order to construct a higher level of abstraction that is hopefully more distinctive and more robust to noise. This extension of RLVC assumes the co-existence of two different kinds of *visual components*:

Primitive (visual) components: They correspond to the individual point features, i.e. to the basic visual features of the set V that were introduced in Chapter 3.[7]

Composite (visual) components: They consist of spatial combinations of lower-level visual components. There is no *a priori* bound on the maximal height of the hierarchy. A composite component can be potentially combined with a primitive component, or with a composite component.

A natural way to represent such a hierarchy is to use a directed acyclic graph $G = (N, E)$, in which each node $v \in N$ corresponds to a visual component, and in which each edge $(v, v') \in E$ models the fact that v' is a part of the composite component v. Thus, G must be *binary*, i.e. any node should have either no child, or exactly two children. The set N_P of the leaves of G corresponds to the set of primitive components, while the set N_C of its internal vertices represents the set of composite components. The resulting perceptual feature space for RLVC is $F_S = N$.

Each primitive component $v_P \in N_P$ is labeled with a visual feature $V(v_P) \in V$ (cf. Definition 3.1). Similarly, each internal node $v_C \in N_C$ is annotated with constraints on the relative position between its parts. In this work, we consider only constraints on the distances between the constituent visual components of the composite components, and we assume that they should be distributed according to a Gaussian law $G(\mu, \sigma)$ of mean μ and standard deviation σ. More precisely, let v_C be a composite component, the parts of which are v_1 and v_2. In order to trigger the detection of v_C in an image $s \in S$, there should be both an occurrence of v_1 and an occurrence of v_2 in s such that their relative Euclidean distance in the image has a sufficient likelihood ν of being generated by a Gaussian of mean $\mu(v_C)$ and standard deviation $\sigma(v_C)$. To ensure symmetry, the location of the composite component is then taken as the midpoint between the locations of v_1 and v_2.

The occurrences of a visual component v in a percept s can accordingly be found using the recursive Algorithm 5.3. A suitable perceptual feature detector $\mathcal{D}_S$ can be derived from this algorithm, by testing whether the resulting set of occurrences contains at least one element:

$$\mathcal{D}_S(s, v) = \textbf{true} \text{ if and only if } \texttt{occurrences}(s, v) \neq \emptyset \qquad (5.9)$$

This perceptual feature detector can be directly embedded inside RLVC.

[7] The terms *primitive components* and *visual features* denote in fact the same entities. The terminology “primitive component” is used to emphasize the fact that we deal with hierarchies of visual features, and not just individual features alone.

[7] Richer constraints could be used, such as taking the relative orientation or the scaling factor between the constituent components into consideration, which would require the use of multivariate Gaussians.

Algorithm 5.3 — Detecting occurrences of visual components

```
$\texttt{occurrences}(s, v)$ :–
  if $v$ is primitive then
    $P \leftarrow \mathcal{D}_L(s)$    {Compute the interest points}
    $O \leftarrow \{\}$
    for $(x, y, w) \in P$ do
      $\boldsymbol{v'} \leftarrow \mathcal{G}_L(s, (x, y, w))$    {Generate the local description}
      if $d(V(v), \boldsymbol{v'}) < \varepsilon$ then
        {The detected visual feature matches the visual component $v$}
        $O \leftarrow O \cup \{(x, y)\}$
      end if
    end for
    return $O$
  else
    $O \leftarrow \{\}$
    $O_1 \leftarrow \texttt{occurrences}(s, \texttt{subcomponent}_1(v))$
    $O_2 \leftarrow \texttt{occurrences}(s, \texttt{subcomponent}_2(v))$
    for all $(x_1, y_1) \in O_1$ and $(x_2, y_2) \in O_2$ do
      $d \leftarrow \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
      if $G(d - \mu(v), \sigma(v)) \geq \nu$ then
        $O \leftarrow O \cup \{((x_1 + x_2)/2, (y_1 + y_2)/2)\}$
      end if
    end for
    return $O$
  end if
```

5.2.3 Closed-Loop Generation of Composite Components

The cornerstone of this extension to RLVC is the way of generating the composite components. The general idea behind our algorithm is to accumulate statistical evidence from the relative positions of the detected visual components in order to find "conspicuous coincidences" of visual components. This is done through an sophisticated perceptual feature generator $\mathcal{G}_S(\{s_1, \ldots, s_n\})$ (cf. Definition 4.5).

Identifying Spatial Relations

We first extract the set F of all the (primitive or composite) components that occur within the set of provided images $\{s_1, \ldots, s_n\}$:

$$F = \{v \in N \mid (\exists i)\ \texttt{occurrences}(s_i, v) \neq \emptyset\}\,. \tag{5.10}$$

We then identify the pairs of visual components the occurrences of which are highly correlated within the set of provided images $\{s_1, \ldots, s_n\}$. This simply amounts to counting the number of co-occurrences for each pair of components in F, then only keeping the pairs the corresponding count of which exceeds a fixed threshold.

Algorithm 5.4 — Generation of composite components

```
$\mathcal{G}_S(\{s_1, \dots, s_n\})$ :-
    $F \leftarrow \{\}$
    for $v \in N$ do
        for $i \leftarrow 1$ to $n$ do
            if occurrences$(s_i, v) \neq \emptyset$ then
                $F \leftarrow F \cup \{v\}$
            end if
        end for
    end for
    $F' \leftarrow \{\}$
    for all $(v_1, v_2) \in F \times F$ do
        if enough co-occurrences of $v_1$ and $v_2$ in $\{s_1, \dots, s_n\}$ then
            $\Lambda \leftarrow \{\}$
            for all $i \in \{1, \dots, n\}$ do
                for all occurrences $(x_1, y_1)$ of $v_1$ in $s_i$ do
                    for all occurrences $(x_2, y_2)$ of $v_2$ in $s_i$ do
                        $\Lambda \leftarrow \Lambda \cup \{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}\}$
                    end for
                end for
            end for
            Apply a clustering algorithm on $\Lambda$
            for each cluster $C = \{d_1, \dots d_m\}$ in $\Lambda$ do
                $\mu \leftarrow$ mean$(C)$
                $\sigma \leftarrow$ stddev$(C)$
                Add to $F'$ a composite component $v_C$ composed of $v_1$ and $v_2$, annotated
                with a mean $\mu$ and a standard deviation $\sigma$
            end for
        end if
    end for
    return $F'$
```

Let now v_1 and v_2 be two components that are highly correlated. A search for a meaningful spatial relationship between v_1 and v_2 is then carried out in the images that contain occurrences of both v_1 and v_2. For each such co-occurrence, we accumulate in a set Λ the distances between the corresponding occurrences of v_1 and v_2. Finally, a clustering algorithm is applied on the distribution Λ in order to detect typical distances between v_1 and v_2. For the purpose of our experiments, we have used *hierarchical clustering* [JMF99]. For each detected cluster, a Gaussian is fitted by estimating a mean value μ and a standard deviation σ. Finally, a new composite component v_C is introduced in the component hierarchy, that has v_1 and v_2 as parts and such that $\mu(v_C) = \mu$ and $\sigma(v_C) = \sigma$. The corresponding piece of code is given in Algorithm 5.4.

Visual Component Validation

Algorithm 5.4 can generate several composite components for a given perceptual class that consists of images (cf. Algorithm 4.1). However, at the end of Algorithm 4.3, at most one generated composite component is to be kept. It is therefore important to notice that the performance of the clustering method is not critical for our purposes. Indeed, irrelevant spatial combinations are automatically discarded, thanks to the variance-reduction criterion of the component selection component. In fact, the reinforcement signal helps direct the search for a good component, which is an advantage over unsupervised methods of building component hierarchies.

5.2.4 Experimental Results

We demonstrate the efficacy of our algorithms on a version of the classical "*car-on-the-hill*" control problem [MA95], where the position and velocity information is presented to the agent visually.

In this episodic task, a car (modeled by a mass point) is riding without friction on a hill, the shape of which is defined by the function:

$$H(p) = \begin{cases} p^2 + p & \text{if } p < 0, \\ p/\sqrt{1+5p^2} & \text{if } p \geq 0. \end{cases}$$

The goal of the agent is to reach as fast as possible the top of the hill, i.e. a location such that $p \geq 1$. At the top of the hill, the agent obtains a reward of 100. The car can thrust left or right with an acceleration of ± 4 Newtons. However, because of gravity, this acceleration is insufficient for the agent to reach the top of the hill by always thrusting toward the right. Rather, the agent has to go left for while, hence acquiring potential energy by going up the left side of the hill, before thrusting rightward. There are two more constraints: The agent is not allowed to reach locations such that $p < -1$, and a velocity greater than 3 in absolute value leads to the destruction of the car.

Formal Definition of the Task

Formally, the set of possible actions is $A = \{-4, 4\}$, while the physical state space is $S = \{(p, s) \mid |p| \leq 1 \wedge |s| \leq 3\}$. The system has the following continuous-time dynamics:

$$\begin{aligned} \dot{p} &= s, \\ \dot{s} &= \frac{a}{M\sqrt{1+\dot{H}(p)^2}} - \frac{g\dot{H}(p)}{1+\dot{H}(p)^2}, \end{aligned}$$

where $a \in A$ is the thrust acceleration, $\dot{H}(p)$ is the first derivative of $H(p)$, $M = 1[kg]$ is the mass of the car, and $g = 9.81[m/s^2]$ is the acceleration due to gravity. These continuous-time dynamics are approximated by the following discrete-time state

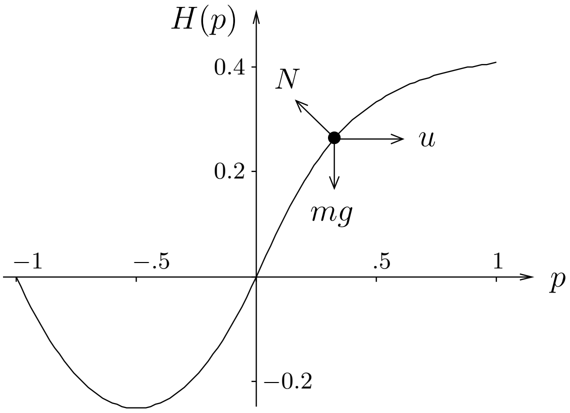


Figure 5.9: The *car-on-the-hill* control problem.

update rule:

$$\begin{aligned} p_{t+1} &= p_t + h\dot{p}_t + h^2\dot{s}_t/2, \\ s_{t+1} &= \dot{p}_t + h\dot{s}_t, \end{aligned}$$

where $h = 0.1$ is the integration time step. The reinforcement signal is defined through this expression:

$$\mathcal{R}((p_t, s_t), a) = \begin{cases} 100 & \text{if } p_{t+1} \geq 1 \wedge |s_{t+1}| \leq 3, \\ 0 & \text{otherwise.} \end{cases}$$

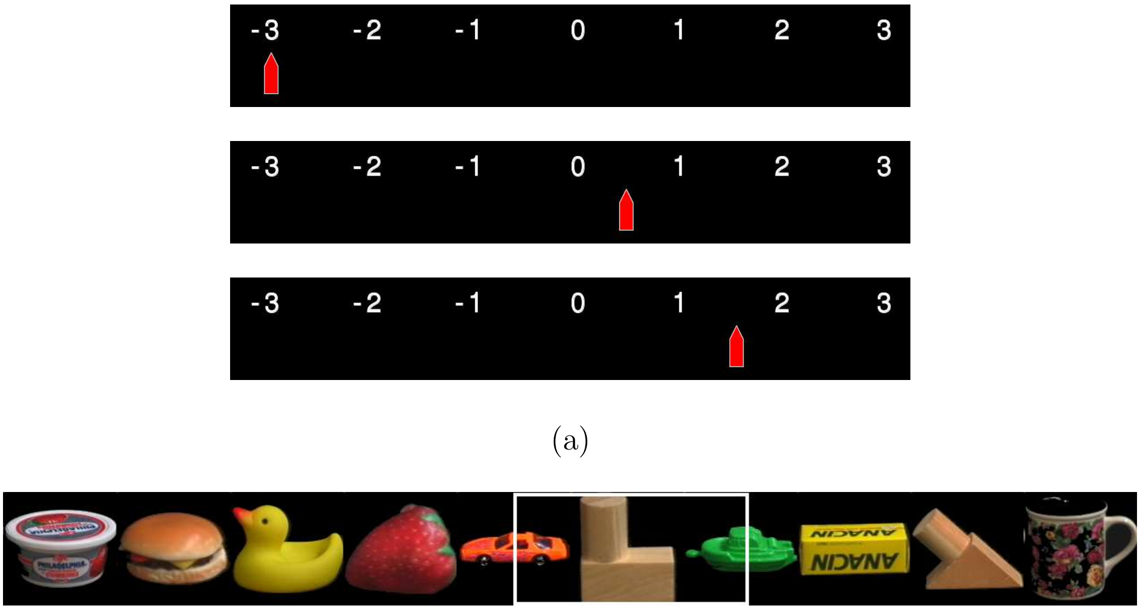
In our setup[8], the discount factor γ was set to 0.75.

Inputs of the Agent

In previous work [MA95, EGW03], the agent was always assumed to have direct access to a numerical measure of its position and velocity. The only exception is Gordon's work in which a visual, low-resolution representation of the global scene is given to the agent [Gor95]. In our experimental setup, the agent is provided with two cameras, one looking at the ground underneath, the second at a velocity gauge. This way, the agent cannot directly know its current position and velocity, but has to suitably interpret its visual inputs to derive them.

Some examples of the pictures the sensors can return are presented in Figure 5.10. The ground is carpeted with a colored banner of 1280×128 pixels that is an assembly of pictures from the COIL-100 database [NNM96]. It is very important to notice

[8] This definition is actually a mix of two coexistent formulations of the car-on-the-hill task [EGW03, MA95]. The major differences with the initial formulation of the problem [MA95] is that the set of possible actions is discrete, and that the goal is at the top of the hill (rather than on a given area of the hill), just like in the definition from Ernst et al. [EGW03]. To ensure the existence of an interesting solution, the velocity is required to remain less than 3 (instead of 2), and the integration time step is set to $h = 0.1$ (instead of 0.01).



(a)

(b)

Figure 5.10: (a) Visual percepts corresponding to pictures of the velocity gauge when $s = -3$, $s = 0.5$ and $s = 1.5$. (b) Visual percepts returned by the position sensor. The region framed with a white rectangle corresponds to the portion of the ground that is returned by the sensor when $p = 0.1$. This portion slides back and forth as the agent moves.

that using individual point features is insufficient for solving this task, since the set of primitive components in the pictures of the velocity gauge are always the same. To know its velocity, the agent has to generate composite components that are sensitive to the distance of the primitive components on the cursor with respect to the primitive components on the digits.

Results

In this experimental setup, we used color differential invariants [GB01] as primitive components. Among all the possible visual inputs (both for the position and the velocity sensors), there were 88 different primitive components. The entire image of the ground includes 142 interest points, whereas the images of the velocity gauge include more or less 20 interest points.

The output of RLVC is a decision tree that defines 157 perceptual classes. Each internal node of this tree tests the presence of one visual component, taken from a set of 91 distinct, highly discriminant features selected by RLVC. Among the 91 selected visual components, there were 56 primitive and 26 composite components. Two examples of composite components that were selected by RLVC are depicted in Figure 5.12. The computation stopped after $k = 38$ refinement steps in Algorithm 4.1.

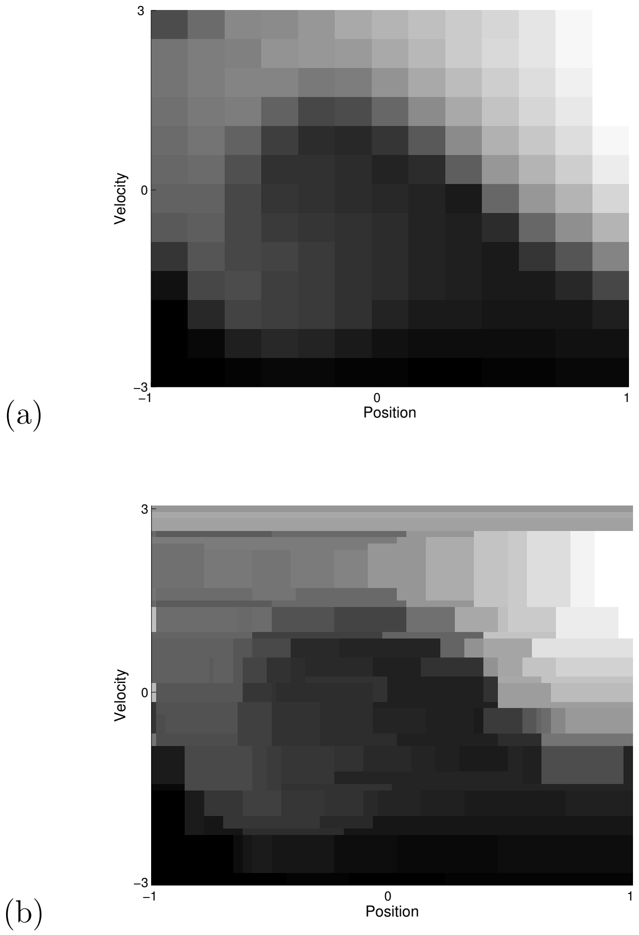


Figure 5.11: (a) The optimal value function, when the agent has a direct access to its current (p, s) state and the input space is discretized in a 13×13 grid. The brighter the location, the greater its value. (b) The value function obtained by RLVC.

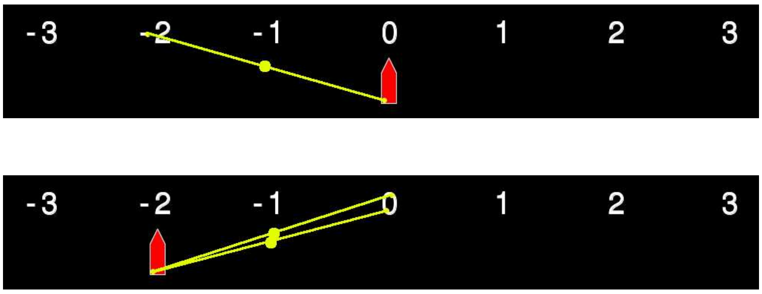


Figure 5.12: Two composite components that were generated, in yellow. The primitive components of which they are composed are marked in yellow. The first component triggers for velocities around 0, whereas the second triggers around -2.

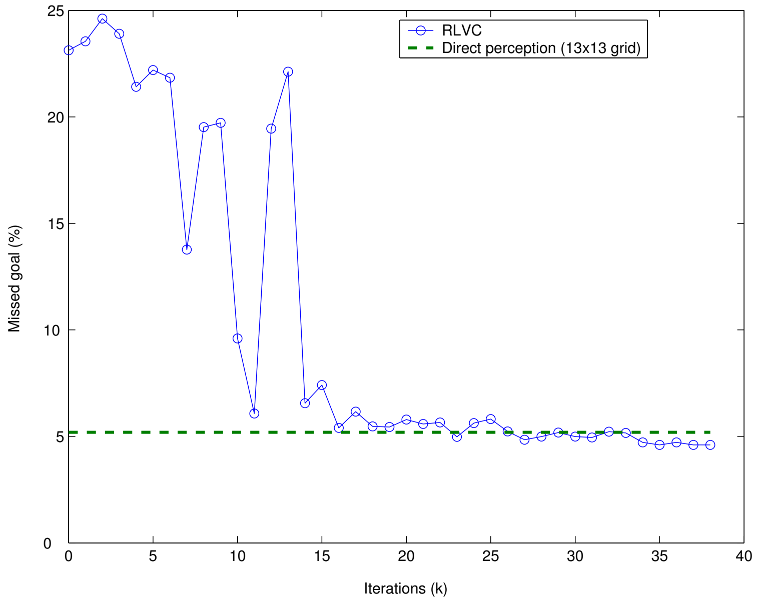


Figure 5.13: Evolution of the number of times the goal was missed over the iterations of RLVC.

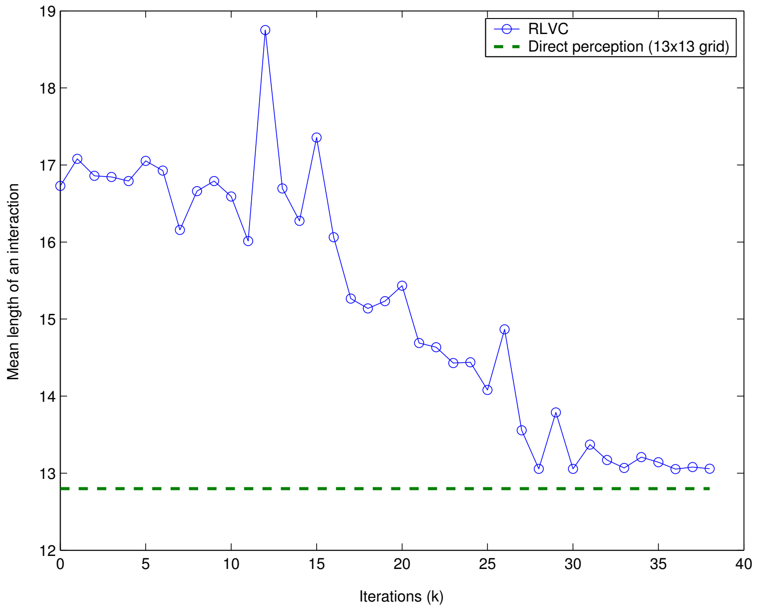


Figure 5.14: Evolution of the mean lengths of the successful trials over the iterations of RLVC.

To show the efficacy of our method, we compare its performance with the scenario in which the agent has a direct perception of its current (p, s) state. In the latter scenario, the state space was discretized in a grid of 13 × 13 cells. The number 13 was chosen since it approximately corresponds to the square root of 157, the number of perceptual classes that were produced by RLVC. This way, RL is provided an equivalent number of perceptual classes in the two scenarios. Figure 5.11 compares the optimal value function of the direct-perception problem with the one obtained through RLVC. Here again, the two pictures are very similar, which indicates the soundness of our approach.

We have also evaluated the performance of the optimal image-to-action mapping

$$\pi^* = \operatorname*{argmax}_{a \in A} Q^*((p, s), a) \tag{5.11}$$

obtained through RLVC. For this purpose, the agent was placed randomly on the hill, with an initial velocity of 0. Then, it used the mapping π^* to choose an action, until it reached a final state. A set of 10,000 such trials were carried out at each step k of Algorithm 4.1. Figure 5.13 compares the proportion of trials that missed the goal (either because of leaving the hill on the left, or because of acquiring a too high velocity) in RLVC and in the direct-perception problem. When k became greater than 27, the proportion of missed trials was always smaller in RLVC than in the direct-perception problem. This advantage in favor of RLVC is due to the adaptive nature of its discretization. Figure 5.14 compares the mean lengths of the successful trials. The mean length of RLVC trials clearly converges to that of the direct-perception trials, while staying slightly larger.

To conclude, RLVC achieves a performance close to the direct-perception scenario. However, the mapping built by RLVC directly links visual percepts to the appropriate actions, without explicitly considering the physical variables.

5.3 Summary

This chapter has proposed two extensions to the basic version of RLVC. The first extension considers techniques for avoiding overfitting in RLVC. The idea is to aggregate visual classes that share similar properties with respect to the theory of Dynamic Programming. Interestingly, this process enhances the generalization abilities of the learned image-to-action mapping, reduces the number of perceptual classes that are built, and progressively eliminates irrelevant perceptual features.

Secondly, an extension of RLVC has been introduced that allows the closed-loop, interactive and task-driven learning of a hierarchy of geometrical combinations of visual features. This is to contrast to most of the prior work on the topic, that uses either a supervised or unsupervised framework [Pia01, FPZ03, BT05, SP06]. Besides the novelty of the approach in the field of computer vision, we have shown its practical value in visual control tasks in which the information provided by the individual point features alone is insufficient for solving the task. Indeed, spatial combinations of visual features are more informative.

CHAPTER

SIX

Function Approximators for Purposive Vision

As shown in the preceding chapters, RLVC succeeds at learning direct image-to-action mapping by selecting highly informative features. One might wonder, however, whether the feature selection process is actually desirable, as it might introduce a high variance in the computed image classifiers, just as in the case of incremental learning of decision trees [GEW06]. Furthermore, selecting visual features requires the introduction of an equivalence relation between the features. This relation is difficult to define. A fixed threshold on a metric in the visual feature space is often used. However, modifying this threshold can lead to significant changes in the learned image-to-action mappings.

Moreover, as argued by Piater [Pia01], both mammalian and state-of-the-art feature-based computer vision systems owe their performance largely to their use of a large number of features. The resulting redundancy or over-completeness creates robustness to various kinds of image variation and degradation. On the contrary, RLVC focuses the attention of the agent only on discriminative visual features. This motivates the design of RL methods that can learn image-to-action mappings by analyzing a large number of raw, partially redundant visual features.

This approach is closely linked to the theory of function approximation in RL, and requires the introduction of sophisticated, general RL algorithms that can be defined independently of a given approximation architecture, such as Fitted Q Iteration *[EGW05] or* Approximate Policy Iteration *[LP03, Mun03].*

6.1 Feature Vectors

RLVC takes advantage of perceptual feature detectors and generators. According to the definitions of Chapter 4, perceptual feature is inherently boolean, in the sense that it is either present or absent given a percept. When dealing with vision-for-action tasks, this is a very coarse abstraction of a visual feature. Indeed, the use of a metric masks the information about the appearance of the visual feature that was extracted by the visual feature generator (cf. Section 3.1.1). In this chapter, we will work with the *raw* visual features by making explicit their appearance. This motivates the following definition:

Definition 6.1. A *raw feature generator* $\mathcal{G}_R : S \mapsto \mathcal{P}(\mathbb{R}^n)$ is a mapping from a percept to a set of real-valued vectors. $\mathbb{R}^n$ is the *raw feature space*, and a vector $\boldsymbol{f} \in \mathbb{R}^n$ is called a *raw feature*.

Any visual feature generator is evidently a raw feature generator, provided that the visual feature space V corresponds to $\mathbb{R}^n$.

Similarly, it is possible to define a raw feature generator for any task in which the perceptual space is continuous (i.e. tasks such that $S = \mathbb{R}^n$, in which a percept is a vector of reals). In the this case, a suitable raw feature generator is simply the identity function:

$$\mathcal{G}_R(\boldsymbol{s}) = \{\boldsymbol{s}\}, \text{ for all } \boldsymbol{s} \in S. \tag{6.1}$$

Note that visual tasks seem at first more complex than continuous tasks, as the raw feature generator of Equation 6.1 is a one-to-one mapping, whereas visual feature generators can produce several vectors of appearance for a single image. All the techniques that will be described in the current chapter apply to any control task such that it is possible to define a raw feature generator on the perceptual space. This includes vision-for-action tasks, as well as any other task with continuously-valued perceptual spaces.

The techniques of this chapter, besides the use of the raw features, are characterized by the use of the *whole set* of raw features. This means that the algorithms below do not take decisions involving only a subset of informative features. This potentially ensures better robustness and generalization abilities.

Of course, as the raw feature space is infinite, it can be sampled only sparsely. As a consequence, we resort to the embedding of function approximators inside the RL process. Among the RL algorithms that use function approximators, we use *Approximate Policy Iteration* (API) [BT96]. Moreover, as raw feature generators are in general one-to-many mappings, *averaging* will be used to deal with multiple raw features. The combination of these two techniques will lead to the definition of *Visual Approximate Policy Iteration* (V-API).

6.2 Related Work

All the RL techniques we have encountered so far represent value functions and state-action value functions as *lookup tables*. As discussed in Chapter 2, these basic techniques converge, provided that some conditions on the parameters of the algorithms are met. However, as argued when introducing RLVC, such algorithms do not scale well with the number of inputs. Indeed, as each cell in a lookup table is uncoupled with the other cells, the number of interactions that are needed to learn an optimal control policy grows with the number of possible percepts. This is the *data dilution* problem. Lookup tables are evidently useless when dealing with infinite perceptual spaces. Furthermore, using lookup tables results in some degree of *redundancy*: If two percepts are similar, they are likely to share similar utilities. This means that the lookup tables can be represented more compactly in memory, hereby reducing the memory requirements. More importantly, this redundancy can be exploited to *generalize* between similar percepts, allowing the agent to reason about percepts not encountered during learning, be it through interpolation or through extrapolation.

As a consequence, current research in reinforcement learning stresses the usage of *function approximators*. The goal is to create an approximation of the entire (state-action) value functions from a finite, limited number of acquired interactions. This is therefore a *regression problem*. Such function approximators can be either parametric or nonparametric, and should follow Occam's razor principle, i.e. they should yield to the simplest explanation of visible information. Function approximators are typically trained by *supervised learning* [Mit97] on a database, so as to minimize an *error function*. Many successful applications of RL have involved the use of function approximators.

Not all function approximation techniques are equally appropriate for reinforcement learning however. Furthermore, there exist multiple ways to embed function approximation inside the reinforcement learning process. In this section, several supervised learning architectures and their application to reinforcement learning are presented. We will consider continuous perceptual spaces: $S = \mathbb{R}^n$. For the time being, only finite action spaces are considered: $A = \{a_1, \ldots, a_m\}$. Infinite action spaces will be considered in Chapter 7.

6.2.1 Linear Approximation Schemes

Function approximators based on a linear combination of basis functions offer a number of advantages that made them particularly successful in the context of reinforcement learning. The general form of linear function approximators for value functions is:

$$\hat{V}(\boldsymbol{s};\boldsymbol{w}) = \sum_{i=1}^{l} w_i \phi_i(\boldsymbol{s}), \tag{6.2}$$

where $\phi_i(\boldsymbol{s}) : S \mapsto \mathbb{R}$ is a set of l *basis functions*, and where $\boldsymbol{w} \in \mathbb{R}^l$ is the *weight vector*. This definition can be extended to state-action value functions as follows:

$$\hat{Q}(\boldsymbol{s}, a;\boldsymbol{w}) = \sum_{i=1}^{l} w_i \phi_i(\boldsymbol{s}, a). \tag{6.3}$$

In this case, the l basis functions are mapping from $S \times A$ to the set of real numbers. The independence between the weight vector and the current percept is the defining characteristic of a linear approximation architecture.

Unlike some more complex function approximation schemes, linear approximators can be easily and efficiently trained through gradient descent. The correct choice of the basis functions is an important factor in determining the success of a linear approximator. Evidently, prior knowledge about the control problem should drive which basis functions are chosen. Several general categories of basis functions are reviewed below, but many variations of these ideas exist.

Grid-Based Discretization

The simplest linear approximation scheme consists in uniformly dividing the perceptual space into a set of hypercubes. There is one basis function $\phi_i(\boldsymbol{s})$ for each hypercube, which is zero everywhere except inside the corresponding hypercube, where it is equal to one. Evidently, the cost of such a discretization is exponential in the number of perceptual dimensions. This is the Bellman curse of dimensionality (cf. Section 4.3) that motivates coarser discretization in areas where little accuracy is needed. A generalization of this idea is to use an arbitrary partitioning of the perceptual space. In general, this does not alleviate the curse of dimensionality, though.

Tile Coding (CMAC)

Instead of using one single grid to approximate the value function, tile coding consists in adding multiple overlapping grids [Alb75]. In tile coding, an exhaustive partitioning of the perceptual space is considered, which is called a *tiling*. Multiple copies of the tiling are superimposed, each offset by a different amount. The tilings need not be uniform grids, an arbitrary partitioning strategy can be used. Tile coding is generally combined with *hashing* techniques to reduce memory requirements. Using tile coding is popular in RL, but its application to high-dimensional problems is not straightforward, as it requires a careful choice of tilings.

Gaussian Kernels

Tile coding uses rectangles with sharp edges, so that strong discontinuities are always produced at the border of the tiles. This is problematic when continuous value functions are to be represented. Therefore, several authors have used Gaussian kernels instead of rectangular kernels, as in *radial basis functions* [Pow87] and in *normalized Gaussian networks* [MD89]. This allows to make gradual, smoother transitions, at the cost of a higher computational expense.

6.2.2 Non-Linear Approximation Schemes

Linear approximators have limited representational power. As a result, many researchers have focused their attention on *non-linear* approximators. Some of them are presented now.

Artificial Neural Networks

By far the most popular non-linear approximator used in reinforcement learning is the *artificial neural network* [RM86]. This approximation architecture has been modeled after the brain, targeting biological plausibility. Artificial neural networks typically consist of many hundreds of simple processing units, the *neurons* which are wired together in a complex communication network. Each unit is a simplified model of a real neuron which fires (sends off a new signal) if it receives a sufficiently strong input signal from the other units to which it is connected. The weights of these connections are adjusted through supervised learning by the *back-propagation algorithm*. Emphasis is generally put on communication networks that form a directed acyclic graph, and, more specifically, on *multilayer perceptrons* [PEL00].

Artificial neural networks constitute a parametric approximation architecture, as the weights suffice to entirely describe the network once its topology is fixed. They have excellent generalization capabilities, especially when the input space is high. On the other hand, the backpropagation algorithm tends to be trapped in a local optimum.

Instance-Based Approximators

Instance-based approximators are not parametric architectures as linear approximators or artificial neural networks: They simply store their training data. When asked to make a prediction, they retrieve the stored points that are closest to the query point according to some distance metric. They then combine these values so as to generate a suitable output. These approximation architectures notably include *nearest neighbor*, *k-nearest neighbor* [CH67] and *locally weighted regression* [AMS97, Sma02]. The computational load of these algorithms is mostly borne during prediction, because learning consists in adding the training data to a database, possibly in a optimized data structure such as a k-d tree [Sam84]. As a consequence, this approach is sometimes known as *lazy learning* algorithms.

6.2.3 Function Approximation in Reinforcement Learning

Several common function approximators have been introduced. We now discuss how these approximation techniques have been embedded inside the reinforcement learning process over years. Many RL algorithms with function approximators were designed from an existing approach in finite-state reinforcement learning, such as Q-learning or Value Iteration. Thus, we cluster these algorithms by the learning architecture for finite spaces they were inspired by. The theoretical properties of many algorithms below are studied in detail by Bertsekas and Tsitsiklis [BT96]. Many pointers to references about the convergence of reinforcement learning with function approximators can be found in Smart's doctoral dissertation [Sma02, Sections 4.2 and 4.3] and in Sutton's RL FAQ [Sut04].

Fitted Temporal-Difference Methods

Generally speaking, the temporal-difference update rules that were described in Chapter 2 can be also exploited when representing state-action value functions through function approximators. This is especially direct if the underlying approximation architecture is *incremental*, to wit, if it is possible to update the approximator by progressively adding newly acquired input-output pairs that associate a state-action pair to an utility. Even if the architecture is *non-incremental* (i.e. if it makes *batch* processing), it is still possible to re-calculate the approximator every time a fixed number of input-output pairs have been computed through the update rules.

Linear approximation schemes have the advantage of *locality*, at the condition that the basis functions are carefully chosen. Indeed, if the basis functions have a non-zero value only in a small area of the perceptual space, a small change in the weight vector $\boldsymbol{w}$ will only have significant impact in a small region only. This is important when dealing with an on-line temporal-difference algorithm, as a global change in the shape of the function is likely to be a source of instability. Evidently, instance-based techniques have also this advantage, but this is not the case of neural networks.

Much research has been devoted to the embedding of function approximators inside the Q-learning process. Watkins has proposed the first work in this line of ideas by using a CMAC [Wat89]. Unfortunately, this method was shown to exhibit divergent behavior in simple grid-world reinforcement learning problems, because Q-learning with linear function approximation is unsound [Bai95]. Sutton has shown that using SARSA along with a CMAC enables more robust and effective learning than Q-learning [Sut96a], but SARSA can also oscillate with linear approximation schemes [TV96]. Later on, the Q-learning update rule has been used in conjunction with radial basis functions [KA97], with instance-based approximators as in *lazy Q-learning* [SS97] and HEDGER [Sma02], as well as with neural networks [Lin91, BT96, tHK00]. Gordon provides a good reference about convergence of various temporal-difference methods [Gor01].

Approximate Actor-Critic Architectures

The use of function approximators is very natural when learning with actor-critic algorithms such as AHC. Recall that such algorithms are based on the evaluation of the value function V^π for a fixed percept-to-action mapping π through temporal-difference learning (cf. Equation 2.53).

Approximate actor-critic architectures are very popular when using a linear parametrization of the value functions as given in Equation 6.2. The most commonly used approach for updating the weight vector $\boldsymbol{w}$ through TD(0) is *stochastic direct gradient* [Wer90], defined by the relation:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_t(\boldsymbol{s_t}, a_t)\left(r_{t+1} + \gamma\hat{V}(\boldsymbol{s_{t+1}}; \boldsymbol{w}) - \hat{V}(\boldsymbol{s_t}; \boldsymbol{w})\right)\nabla_{\boldsymbol{w}}\hat{V}(\boldsymbol{s_t}; \boldsymbol{w}). \tag{6.4}$$

Therefore, the direct gradient algorithm assumes that using the partial derivative of the temporal-difference error with respect to the current percept will result in the correct solution if it converges. However, when the parameters $\boldsymbol{w}$ are updated, the desired output changes too. This might result in oscillations or worse in divergence, even for very small Markov decision processes [Bai95, TV96]. Baird uses an evolution of this rule [Bai95], by combining the partial derivative of the utility of the *current* *state* with that of the *next state*:

$$\begin{aligned}\boldsymbol{w} \quad \leftarrow \quad & \boldsymbol{w} + \alpha_t(\boldsymbol{s_t}, a_t)\left(r_{t+1} + \gamma \hat{V}(\boldsymbol{s_{t+1}}; \boldsymbol{w}) - \hat{V}(\boldsymbol{s_t}; \boldsymbol{w})\right) \\ & \left(\nabla_{\boldsymbol{w}} \hat{V}(\boldsymbol{s_t}; \boldsymbol{w}) - \gamma \, \nabla_{\boldsymbol{w}} \hat{V}(\boldsymbol{s_{t+1}}; \boldsymbol{w})\right). \qquad (6.5)\end{aligned}$$

This update rule is known as *stochastic residual gradient*. These two update rules are based on TD(0). The use of eligibility traces leads to approximated versions of TD(λ), the very positive theoretical properties of which are discussed by Tsitsiklis and Van Roy [TV97].

Two variants of TD(λ) with linear approximation schemes replace the local gradient descent by a direct resolution of systems of linear equations. These so-called *least-squares methods* are both based on iterative matrix updates, and exhibit better convergence rates. The first method is called *Least-Squares TD Learning* (LSTD(λ)) [Boy02]. Its basic version, called LSTD [BB96], does not use eligibility traces. Convergence of LSTD(λ) is discussed in recent work [NB03, Sch03]. A second least-squares method has been recently introduced under the name of *Least-Squares Policy Evaluation* (LSPE) [NB03].

Approximate actor-critic architectures have also been often combined with neural networks. The early work by Anderson uses an artificial neural network [And87], and succeeds at solving the pole-balancing problem. Similarly, Lin has embedded neural network approximations with both the AHC and the Q-learning algorithm [Lin91]. Zhang and Dietterich learned strategies for job-shop scheduling through TD(λ) and neural networks [ZD95]. Probably one the greatest successes of RL has been the *TD-Gammon algorithm* that turns a computer into an excellent Backgammon player [Tes95]. Similarly to Zhang and Dietterich, TD-Gammon combines TD(λ) and neural networks, the inputs of which are hand-coded expert features. Note that the effectiveness of this approach strongly depends on the quality of the input features. More recently, Coulom has proposed the *continuous TD*(λ) algorithm that can handle complex, continuous motor-control problems [Cou02].

Fitted Value Iteration

Much recent research in reinforcement learning focuses on embedding function approximators inside the Value Iteration process (cf. Section 2.2.4). This general, increasingly popular family of RL methods is known as *Fitted Value Iteration* [Gor95, MS06]. Fitted Value Iteration can be written as a stochastic iterative rule [SM05], which makes it a special form of *Approximate Value Iteration* [TV96].

The theoretical properties of Fitted and Approximate Value Iteration in the context of dynamic programming are discussed by Bertsekas and Tsitsiklis [BT96]. These results were later extended by Gordon [Gor99], who showed that the use of a particular family of function approximators (the *averagers*) ensures the convergence of Fitted Value Iteration. Unfortunately, the convergence properties from Bertsekas and Tsitsiklis are defined with respect to the maximum norm (cf. Section 2.2.2), but approximation architectures generally strive at minimizing the errors in the

Euclidean norm. This has motivated the study of convergence properties of Fitted Value Iteration in other norms by Munos [Mun05, Mun06a].

Very sketchily, Fitted Value Iteration methods are characterized by two main properties: (1) the way Bellman backup operators are approximated[1], and (2) the family of function approximators that is used. As discussed by Munos, there exist at least two possible approaches for approximating the Bellman backup operators in the framework of reinforcement learning [Mun06a, Section 6]: either through a generative model of the underlying MDP, which leads to *Sampling-Based Fitted Value Iteration* [SM05, MS06]; or through a stochastic version of the Bellman backup operators, which leads to the *Fitted Q Iteration* algorithm [EGW05] (cf. also Section 6.4.2). The exploited function approximator scheme must be carefully chosen, as already noticed by Boyan and Moore in the early *grow-support algorithm* [BM95]: Successfully investigated families of function approximators include kernels [OS02], instance-based approximators [EGW05], regression trees [EGW05], as well as neural networks [Rie05].

Direct Policy Search

Finally, we also note the existence of the *direct policy search methods* that are related to function approximation techniques. Such methods search for optima in the space of all possible percept-to-action mappings by directly examining different policy parametrizations, but they bypass the computation of the value functions. These methods are especially useful for partially observable control problems, but are out of the scope of this dissertation. The interested reader can find a review of state-of-the-art direct policy search methods in Peshkin's doctoral dissertation [Pes02].

6.3 Approximate Policy Iteration

Approximate Policy Iteration (API) is a generalization of Modified Policy Iteration (cf. Section 2.2.5) to arbitrary perceptual spaces S, that are possibly continuous. Therefore, API is an alternative to Fitted Value Iteration, in which Modified Policy Iteration plays the same role as Value Iteration. In Chapter 2, we have introduced actor-critic architectures as reinforcement learning methods that iteratively improve a percept-to-action mapping by separately storing the current policy itself and its value function, and by using a stochastic version of Bellman backup operators.

Actor-critic architectures contrast with Modified Policy Iteration that can evaluate the current percept-to-action mapping π_k by repeatedly applying the exact Bellman backup operator H^{π_k}. This is possible because Modified Policy Iteration is defined in the dynamic programming framework. Nevertheless, it was the designer's responsibility to fix how the policy π_k is represented. Two equivalent representation systems are possible: (1) either directly through a table that maps percepts to actions, which was possible since the perceptual space was assumed finite, or (2)

[1] Indeed, the operators of Sections 2.2.2 and 2.2.3 are only applicable to *finite* state-action spaces for which a model of the environment is known.

implicitly from the current state-action value function by extracting a greedy policy from this function:

$$\pi_k(s) = \operatorname*{argmax}_{a \in A} Q^{\pi_{k-1}}(s, a). \tag{6.6}$$

In infinite perceptual spaces, the difference between these two approaches becomes less evident, as the Bellman backup operator can only be approximated, and as it is impossible to directly represent the current percept-to-action mapping by table lookup. In this dissertation, we will therefore distinguish between the following two architectures:

- *Approximate actor-critic architectures.* In this setup, the current percept-to-action mapping π_k and its value function $V^{\pi_k}(s)$ or its state-action value function $Q^{\pi_k}(s, a)$ are separately stored. Approximate actor-critic architectures have been discussed above.

- *Approximate Policy Iteration architectures.* In this case, only state-action value functions are stored, and the current percept-to-action mapping is defined as the greedy policy with respect to this state-action value function. In such architectures, the value functions $V^{\pi_k}(s)$ are not relevant: It is impossible to directly extract a greedy percept-to-action mapping from them, because this operation would require an exact model of the environment. Approximate Policy Iteration architectures are the topic of this chapter.

In both cases, approximate versions of Bellman backup operators are used. Note that the terminology "Approximate Policy Iteration" is somewhat misleading, as it has often been used in the literature to designate approximate actor-critic architectures, in the sense that has just been defined. In fact, Approximate Policy Iteration can be thought of as a particular class of approximate actor-critic methods, in which the actor is implicitly deduced from the critic component. As a consequence, the policy evaluation and the policy improvement steps are essentially blended into a single process, since there is no need for an intermediate, complete representation of the generated percept-to-action mappings.

Bertsekas and Tsitsiklis' reference book [BT96, Section 6.2] discusses several results about the convergence of Approximate Policy Iteration under the maximum norm. Munos generalizes these convergence results to quadratic norms [Mun03] in the context of linear approximation architectures. As argued by Munos [Mun03] as well as by Lagoudakis and Parr [LP03], Approximate Policy Iteration has the empirical advantage of quick convergence over Value Iteration. This motivates the use of API for purposive vision.

6.3.1 Nonparametric Approximate Policy Iteration

Contrarily to the case of Fitted Value Iteration, there exist only a few concrete algorithms that apply Approximate Policy Iteration in the sense defined above. To the best of our knowledge, API has only been used in conjunction with linear approximation architectures, leading to the *Least-Squares Policy Iteration* (LSPI) algorithm [LP03]. In LSPI, the evaluation of a greedy percept-to-action mapping, so as to obtain its approximated state-action value function using samples, is carried out by the LSTDQ algorithm [LP03]. LSTDQ is a variation of the LSTD algorithm for approximate actor-critic architectures [BB96].

Although they are very useful for physical control problems, linear approximation schemes are not well suited to purposive vision tasks, as their representational power is limited. In the next sections, a general, nonparametric RL version of Approximate Policy Iteration is described. More precisely, we introduce the *Nonparametric Approximate Policy Iteration* (Nonparametric API) algorithm that has the two following important properties:

- This algorithm can accommodate any function approximation architecture, such as instance-based methods, neural networks or regression trees. This contrasts with LSPI that explicitly targets parametric, linear approximation schemes. In fact, Nonparametric API underlines the black-box aspect of supervised learning algorithms, and does not make any assumption about this algorithm. This is a point of view that is shared with the Fitted Q Iteration algorithm by Ernst et al. [EGW05].
- Nonparametric API learns near-optimal percept-to-action mappings from a static database of interactions, and does not require a model of the surrounding MDP. Just like Least-Squares Policy Iteration, Nonparametric API is a fully off-policy algorithm and can, in principle, use data collected from any reasonable sampling distribution [LP03].

In Section 6.5, Nonparametric API will be combined with a function approximator that is specifically tuned to problems that admit a raw feature generator (cf. Definition 6.1). However, Nonparametric API can easily be used with another function approximation architecture. This motivates the following definition:

Definition 6.2. A *state-action value function approximator* is a mapping $\hat{Q} : S \times A \mapsto \mathbb{R}$ that assigns a real number to each state-action pair. The family $\mathcal{F}$ designates the set of all state-action value function approximators that can be represented with the considered approximation scheme.

Because we deal with black-box learning procedures, the existence of a learning oracle $\mathcal{L}$ is assumed. This oracle returns a state-action value function approximator from a database:

Definition 6.3. Let $D = \{\langle s_t, a_t, v_t \rangle\}$ be a finite database of triples that are made up of a state $s_t \in S$, an action $a_t \in A$ and a utility $v_t \in \mathbb{R}$. A *learning oracle* is a function:

$$\mathcal{L} : \mathcal{P}(\mathcal{S} \times \mathcal{A} \times \mathbb{R}) \mapsto \mathcal{F}, \tag{6.7}$$

that builds a function approximator $\hat{Q} = \mathcal{L}(D)$ that is the closest possible to the given sample distribution, in the sense that it minimizes a given error function over the family $\mathcal{F}$ with respect to the database D.

Algorithm 6.1 — Nonparametric Approximate Policy Iteration

```
approximate-policy-iteration $(\{\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle\})$ :-
    $k \leftarrow 1$
    $\hat{Q}^{(1)} \leftarrow \mathcal{L}\left(\{\langle s_t, a_t, r_{t+1}\rangle\}\right)$
    loop
        $\hat{Q}^{(k+1)} \leftarrow$ greedy-evaluation $\left(\hat{Q}^{(k)}, \{\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle\}\right)$
        if equivalent-approximators $\left(\hat{Q}^{(k)}, \hat{Q}^{(k+1)}, \{\langle s_t, a_t\rangle\}\right)$ then
            return $\pi[\hat{Q}^{(k)}]$
        end if
        $k \leftarrow k + 1$
    end loop
```

6.3.2 Implicit Representation of the Generated Policies

As mentioned above, Nonparametric API relies on a simple, state-of-the-art principle that was already exploited in Least-Squares Policy Iteration [LP03]: Any state-action value function $Q(s, a)$ induces a greedy percept-to-action mapping $\pi[Q]$ that always selects the action maximizing Q:

$$\pi[Q]\,(s) = \operatorname*{argmax}_{a \in A} Q(s, a), \ \text{ for each } s \in S. \tag{6.8}$$

Note that the maximization of this equation can easily be achieved, as the action space is assumed finite.

As a consequence, there is no need of an explicit, separate representation system for policies if we keep track of the state-action value functions[2]. Thus, it is sufficient for Nonparametric API to generate a sequence of state-action value function approximators $\hat{Q}^{(1)}, \hat{Q}^{(2)}, \hat{Q}^{(3)}, \ldots$ Each approximator $\hat{Q}^{(k)}$ in this sequence implicitly represent a greedy percept-to-action mapping π_{k+1} that can be evaluated on demand:

$$\pi_{k+1}(s) = \pi[\hat{Q}^{(k)}]\,(s) = \operatorname*{argmax}_{a \in A} Q^{(k)}(s, a), \ \text{ for each } s \in S. \tag{6.9}$$

6.3.3 Main Algorithm

Recall that Modified Policy Iteration relies on two interleaved processes: Modified Policy Evaluation and Policy Improvement. Thanks to the implicit representation of the policies, the Policy Improvement step of Nonparametric API becomes trivial: It is indeed sufficient to define $\hat{Q}^{(k+1)}$ as the state-action value function of the greedy policy with respect to $\hat{Q}^{(k)}$. This is precisely the result of the Modified Policy Evaluation component that will be described in the next section. Algorithm 6.1 summarizes the backbone of Nonparametric API. The input to the algorithm is

[2] In terms of the notation of Section 2.2.5, this would correspond to a version of Modified Policy Iteration algorithm that would not not keep track of π_k, but only of Q^{π_k}.

Algorithm 6.2 — Comparison of two state-action value function approximators

```
equivalent-approximators $\left(\hat{Q}, \hat{Q}', \{\langle s_t, a_t\rangle\}\right)$ :–
    $d \leftarrow 0$
    for $t \leftarrow 1$ to $T$ do
        $q \leftarrow \hat{Q}(s_t, a_t)$
        $q' \leftarrow \hat{Q}'(s_t, a_t)$
        $d \leftarrow d + (q - q')^2$
    end for
    if $d < \varepsilon$ then
        return true
    else
        return false
    end if
```

a database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$, which makes of Nonparametric API an off-policy, model-free RL algorithm. The third line implicitly builds an initial percept-to-action mapping π_1 that maximizes immediate reinforcements.

The algorithm stops when the difference between two successive $\hat{Q}^{\pi_k}$ becomes negligible. Unfortunately, because the state-action value function approximators are not lookup tables, it is impossible to directly compare them. To this end, we rather sample their values for the state-action pairs that are present in the input database of interactions. This is a natural choice, because the learning oracle $\mathcal{L}$ builds the approximators from utilities sampled at these control points. Then, the test of equivalence consists in testing whether the *empirical loss* on the utilities drops below a threshold ε:

$$\frac{1}{T}\sum_{t=1}^{T} l\left(\hat{Q}^{(k)}(s_t, a_t) - \hat{Q}^{(k+1)}(s_t, a_t)\right) < \varepsilon \tag{6.10}$$

where T is the number of interactions in the database, and where the loss function l is usually an absolute or quadratic function [Mun06a, Section 3]. In practice, we have been using the quadratic loss function in our experiments. The corresponding piece of code is given in Algorithm 6.2.

6.3.4 Modified Policy Evaluation in Nonparametric API

We now describe a Modified Policy Evaluation component that is suited to Nonparametric API. This component is to be plugged as `greedy-evaluation` inside Algorithm 6.1. Its role consists in computing the state-action value function $\hat{Q}^{(k+1)}$ of a percept-to-action mapping $\pi_{k+1} = \pi[\hat{Q}^{(k)}]$ that is greedy with respect to $\hat{Q}^{(k)}$, given the input database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$.

To this end, similarly to the original version of Modified Policy Evaluation, a sequence of state-action value function approximators $\hat{Q}_i$ is generated. This is done according to the principle of Modified Policy Iteration, to wit, $\hat{Q}_{i+1}$ is defined as

Algorithm 6.3 — Evaluation of a greedy policy in Nonparametric API

```
greedy-evaluation $\left(\hat{Q}^{(k)}, \{\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle\}\right)$ :-
  $i \leftarrow 1$
  $\hat{Q}_1 \leftarrow Q^{(k)}$
  {The decisions of the policy $\pi_{k+1}$ are cached in an array $c[t+1]$}
  for $t \leftarrow 1$ to $T$ do
    $c[t+1] \leftarrow \operatorname{argmax}_{a' \in A} \hat{Q}^{(k)}(s_{t+1}, a')$
  end for
  loop
    $D \leftarrow \{\}$
    for $t \leftarrow 1$ to $T$ do
      $D \leftarrow D \cup \left\{\left\langle s_t, a_t, r_{t+1} + \gamma\ \hat{Q}_i\left(s_{t+1}, c[t+1]\right)\right\rangle\right\}$
    end for
    $\hat{Q}_{i+1} \leftarrow \mathcal{L}(D)$
    if equivalent-approximators $\left(\hat{Q}_i, \hat{Q}_{i+1}, \{\langle s_t, a_t\rangle\}\right)$ then
      return $\hat{Q}_i$
    end if
    $i \leftarrow i + 1$
  end loop
```

$(H^{\pi_{k+1}}\hat{Q}_i)$, where $H^{\pi_{k+1}}$ is the Bellman backup operator that corresponds to π_{k+1}. However, there are two differences with respect to finite perceptual spaces: (1) The functions $\hat{Q}_i$ are now approximations that are built through the learning oracle $\mathcal{L}$, and (2) the Bellman backup operator for the percept-to-action mapping π_{k+1} cannot be directly used anymore because the perceptual space is possibly infinite. To solve this problem, let us consider the Bellman backup operator for π_{k+1}. According to Equations 2.27 and 6.9, it is defined as:

$$\begin{aligned}(H^{\pi_{k+1}}\hat{Q})(s,a) &= \mathcal{R}(s,a) + \gamma \int_{s' \in S} \mathcal{T}(s,a,s') \sum_{a' \in A} \pi_{k+1}(s',a')\ \hat{Q}(s',a') &(6.11)\\ &= \mathcal{R}(s,a) + \gamma \int_{s' \in S} \mathcal{T}(s,a,s')\ \hat{Q}\left(s', \operatorname*{argmax}_{a' \in A} \hat{Q}^{(k)}(s',a')\right). &(6.12)\end{aligned}$$

As the dynamics of the system are unknown, Nonparametric API uses the stochastic version of this equation for an interaction $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$, which can be motivated similarly to the update rule of Q-learning (cf. Section 2.4.2):

$$r_{t+1} + \gamma\ \hat{Q}\left(s_{t+1}, \operatorname*{argmax}_{a' \in A} \hat{Q}^{(k)}(s_{t+1}, a')\right). \tag{6.13}$$

Thus, the sequence of state-action value function approximators that are generated by the approximate version of Modified Policy Evaluation component is defined through the following batch mode recurrence:

$$\hat{Q}_{i+1} \leftarrow \mathcal{L}\left(\left\{\left\langle s_t,\ a_t,\ r_{t+1} + \gamma\, \hat{Q}_i\left(s_{t+1}, \operatorname*{argmax}_{a' \in A} \hat{Q}^{(k)}(s_{t+1}, a')\right)\right\rangle\right\}\right) . \qquad (6.14)$$

A new state-action value approximator is constructed by the learning oracle $\mathcal{L}$ by each application of this update rule. This rule can be motivated similarly to Fitted Q Iteration [EGW05]: The stochastic aspect of the environment will eventually be captured by the function approximators. Interestingly, all the elements

$$c_{t+1} = \operatorname*{argmax}_{a' \in A} \hat{Q}^{(k)}(s_{t+1}, a') \qquad (6.15)$$

that are used in Equation 6.14 are constant across all the applications of the update rule. Therefore, it is possible to cache these values c_{t+1} in an array in order to speed up the computations.

Similarly to the main algorithm, this learning process stops when the difference between two successive $\hat{Q}_i$ drops below a threshold. The first element $\hat{Q}_1$ of the sequence can be chosen freely. In practice, it is set to $\hat{Q}^{(k)}$: This is a starting point that reduces the number of iterations before convergence, as the percept-to-action mapping π_{k+1} generally shares common decisions with π_k. The resulting code is outlined in Algorithm 6.3.

This concludes the general definition of Nonparametric API. In the next sections, we will discuss function approximators that are suited when a raw feature generator is available.

6.4 Extremely Randomized Trees

This section introduces *Regression Extra-Trees*[3], a family of nonparametric function approximators that results from research at the University of Liège [GEW06]. We restrict our study to the case where all attributes (both the inputs and the output) are numerical, hence the name "Regression" Extra-Trees. Thus, a Regression Extra-Tree model defines a mapping $\mathbb{R}^n \mapsto \mathbb{R}$, and is learned from a database of real-valued input-output pairs $\langle \boldsymbol{x_i}, y_i \rangle \in \mathbb{R}^n \times \mathbb{R}$.

As their name indicates, Regression Extra-Trees are tree-based function approximators. Extra-Trees are a highly successful evolution of CART trees [BFS84], of Bagging [Bre96], and of Random Forests [Bre01]. Extra-Trees are characterized by a strong random component that is present during their generation, which leads to a strong reduction in both bias and variance of the function approximators. Extra-Trees also have excellent performance in generalization [GEW06].

6.4.1 Extra-Tree Induction

Internally, an Extra-Tree model is constituted by a forest of M independent decision trees. Each of their internal nodes is labeled by a threshold on one of the

[3] The name "Extra-Trees" stands for *EXTremely RAndomized Trees*.

Algorithm 6.4 — General structure for Regression Extra-Tree learning

```
extra-trees($\{\langle \boldsymbol{x_i}, y_i\rangle\}, M$) :-
    $\mathcal{T} \leftarrow \phi$
    for $i \leftarrow 1$ to $M$ do
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{$subtree($\{\langle \boldsymbol{x_i}, y_i\rangle\}$)$\}$
    end for
    return $\mathcal{T}$
```

Algorithm 6.5 — Recursive induction of a single subtree

```
subtree($\{\langle \boldsymbol{x_i}, y_i\rangle\}$) :-
    if too few samples or each input $\boldsymbol{x_i}$ is constant or $y_i$ is constant then
        $o \leftarrow \text{mean}(\{y_i\})$
        return a leaf labeled with output $o$
    else
        for $v \leftarrow 1$ to $n$ do
            $a, b \leftarrow \min_i\{x_{i,v}\}, \max_i\{x_{i,v}\}$
            $t[v] \leftarrow$ random value in $[a, b]$
            $s[v] \leftarrow$ score($\{\langle \boldsymbol{x_i}, y_i\rangle\}, v, t[v]$)
        end for
        $v^* \leftarrow \text{argmax}_v\{s[v]\}$
        $i_\ominus \leftarrow \{i \mid x_{i,v*} < t[v^*]\}$
        $i_\oplus \leftarrow \{i \mid x_{i,v*} \geq t[v^*]\}$
        $T_\ominus \leftarrow$ subtree($\{\langle \boldsymbol{x_i}, y_i\rangle\} \mid i \in i_\ominus$)
        $T_\oplus \leftarrow$ subtree($\{\langle \boldsymbol{x_i}, y_i\rangle\} \mid i \in i_\oplus$)
        return a binary decision node $\langle t[v^*], T_\ominus, T_\oplus\rangle$
    end if
```

input attributes that is to be tested in that node. The leaves are labeled by a regression output. The regression response for an input $\boldsymbol{x}$ is obtained by computing the response of each subtree. This is achieved by starting at the root node, then progressing down the tree according to the result of the thresholding tests found during the descent, until a leaf is reached. By doing so, each subtree votes for a regression output. Finally, the mean of these outputs is assigned to the sample.

Algorithms 6.4 and 6.5 describe how to build a Regression Extra-Tree model. In this pseudo-code, $\boldsymbol{x_i} \in \mathbb{R}^n$ contains the input attributes of the ith sample in the learning set and $y_i \in \mathbb{R}$ is the observed regression output for this sample. We assume the existence of a function `score`($\{\langle \boldsymbol{x_i}, y_i\rangle\}, v, t$) that returns the score of the threshold t on the vth input component with respect to the database $\{\langle \boldsymbol{x_i}, y_i\rangle\}$. In our implementation, variance reduction was used as the `score` function (cf. Section 4.4.3). These algorithms are identical to those presented by Geurts et al. [GEW06] and are restated here in order to complement the description of our distributed algorithms (cf. Section 6.6). We refer the reader to Geurts et al. [GEW06] for a complete and thorough treatment.

6.4.2 Applications of Extra-Trees

The choice of exploiting Extra-Trees to deal with vision-for-action problems is motivated by their good performance in supervised learning problems [GEW06], but also by previous applications in the domains of computer vision [MGPW05, Mar05] and reinforcement learning [EGW05]. These positive results are described below. Our definition of Visual Approximate Policy Iteration has been inspired by some of the ideas from these applications.

Image Classification

The *image classification* task is a common benchmark for supervised learning algorithms in computer vision. This problem consists in attributing a class number to the image of an object, so that all the images of a given object are assigned with the same class number, and so that no image of an alien object is associated with this class. The image classifier is trained by a database of pairs mapping reference images to the corresponding class. Marée et al. [Mar05, MGPW05] have shown that Classification Extra-Trees can be used to reach state-of-the-art performance on this task using the value of raw pixels (see also Section 3.4).

In practice, they have proposed to exploit *Classification* Extra-Tree models that define a mapping $\mathbb{R}^n \mapsto C$, where $C = \{c_1, \ldots, c_m\}$ is a set of object classes. The induction of such classification models is similar, but not identical, to Algorithm 6.4 [GEW06]. The models are now generated from a database of input-output pairs $\langle \boldsymbol{x}_i, c_i \rangle \in \mathbb{R}^n \times C$. Marée et al. assume the presence of a visual feature generator $\mathcal{G}_V$ that extracts a set of visual features from any image $s \in S$. Their research work has mainly focused on the extraction of image patches at random interest points, but any other visual feature generator could be used as well.

Evidently, Classification Extra-Trees cannot be used directly to classify an image, because a visual feature generator maps *one* visual percept to *many* visual features (cf. Definition 3.1). This fundamental problem is solved by applying the Extra-Tree model independently on each visual feature that is present in the input image. This process generates one classification output per visual feature in the input image. Then, the output of the function approximator is defined as the class that obtains most votes among these classification outputs. This approach is depicted in Figure 6.1.

We now describe how the corresponding Classification Extra-Tree model is generated. Given a training database of image-class pairs $\langle s_i, c_i \rangle \in S \times C$, a second database is constructed as follows:

$$\{\langle \boldsymbol{v}, c \rangle \mid (\exists i)\ (\exists \boldsymbol{v'} \in \mathcal{G}_V(s_i))\ \boldsymbol{v} = \boldsymbol{v'} \text{ and } c = c_i\}\,. \tag{6.16}$$

Intuitively, this second database contains one entry for each visual feature in each training image. Each of these features is associated with the class number of the object whose image has triggered the detection of the visual feature. The entries of this second database are elements of $\mathbb{R}^n \times C$, and are used to train the Classification Extra-Tree model.

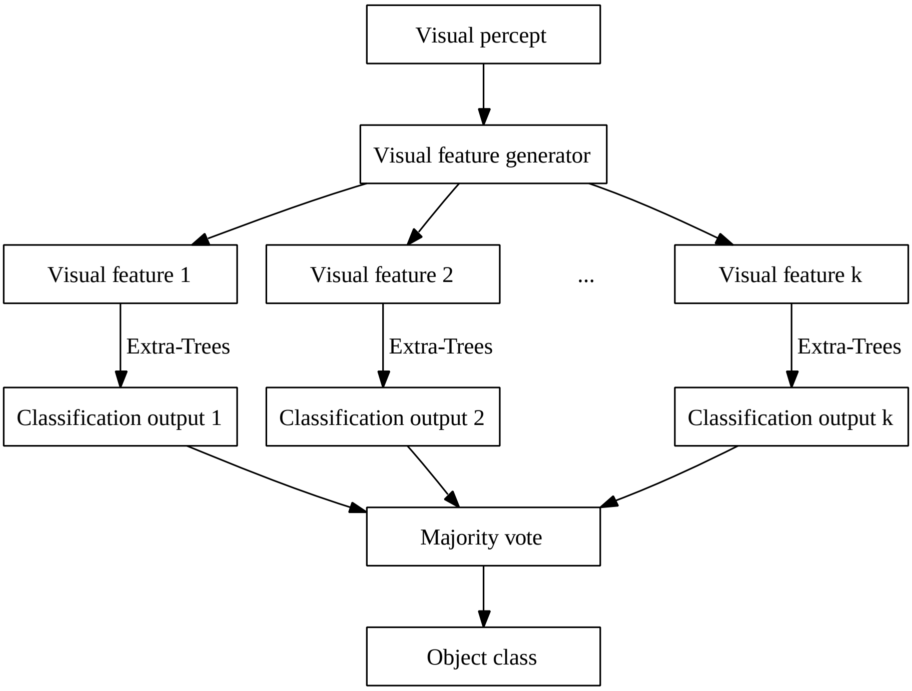


Figure 6.1: Assigning a class to an image using a Classification Extra-Tree model.

Tree-Based Batch Mode Reinforcement Learning

A second successful application of the Extra-Trees have been to use them as function approximators for reinforcement learning. Ernst et al. [EGW05] have proposed to embed Regression Extra-Trees in the Fitted Value Iteration process (cf. Section 6.2.3). Just like Nonparametric API, *Fitted Q Iteration* is a model-free RL algorithm that takes as input a database of interactions $\langle \boldsymbol{s_t}, a_t, r_{t+1}, \boldsymbol{s_{t+1}} \rangle$. The perceptual space is usually assumed continuous (i.e. $S = \mathbb{R}^n$). Similarly to Nonparametric API, the Fitted Q Iteration algorithm makes a batch treatment of the stochastic version of a Bellman backup operator. Nevertheless, Fitted Q Iteration considers the Bellman backup operator H for the optimal state-action value function (cf. Equation 2.27), instead of the operator H^π for a fixed policy π. Thus, Fitted Q Iteration directly targets the evaluation of the optimal state-action value function Q^*, whereas Nonparametric API evaluates the state-action value functions of a sequence of improving policies.

Fitted Q Iteration generates a sequence of state-action value function approximators $\hat{Q}^{(i)}(\boldsymbol{s}, a)$ that converge to $Q^*(\boldsymbol{s}, a)$. Each of these approximators is represented through Regression Extra-Trees. However, because the action space is discrete, the actions cannot be given as an input to a Regression Extra-Tree model[4]. The solution to this problem is straightforward: The single Extra-Tree model $\hat{Q}^{(i)}(\boldsymbol{s}, a)$ is replaced by $|A|$ Extra-Tree models $\hat{Q}_a^{(i)}(\boldsymbol{s})$, one for each possible action $a \in A$, and

[4] The full definition of Extra-Trees by Geurts et al. [GEW06] does not include this limitation. Symbolic variables can be used simultaneously to numeric variables. This restriction has been introduced in this dissertation for teaching purposes.

the following rule is used to evaluate the function:

$$\hat{Q}^{(i)}(\boldsymbol{s}, a) = \hat{Q}_a^{(i)}(\boldsymbol{s}), \tag{6.17}$$

for each percept $s \in S$ and each action $a \in A$. This discussion allows us to formulate the recurrence rule of Fitted Q Iteration as follows:

$$\begin{aligned} \hat{Q}_a^{(i)}(\boldsymbol{s}) &= \texttt{extra-trees}\left\{\langle \boldsymbol{s_t}, r_{t+1}\rangle \mid a_t = a\right\}, & (6.18)\\ \hat{Q}_a^{(i+1)}(\boldsymbol{s}) &= \texttt{extra-trees}\left\{\left\langle \boldsymbol{s_t}, r_{t+1} + \gamma \operatorname*{argmax}_{a' \in A} \hat{Q}_a^{(i)}(\boldsymbol{s_{t+1}})\right\rangle \mid a_t = a\right\}, & (6.19)\end{aligned}$$

where `extra-trees` designates Algorithm 6.4. Intuitively speaking, Fitted Q Iteration turns the Q-learning algorithm into a sequence of supervised regression problems.

Application to Real-Time Human Silhouette Detection

Finally, we also point out the use of Classification Extra-Trees in the framework of real-time silhouette classification in a video sequence [BJV06, Bar06]. This approach uses background subtraction techniques so as to extract the moving silhouettes in a video stream. The system then classifies these silhouettes as human or non-human according to supervised learning on a novel kind of morphological features that are borrowed from structured-based vision (cf. Section 3.1.2). The classification can be run in real-time, as the application of an Extra-Tree model is a very quick operation. We mention this work here, as it is one of our personal contributions that is not directly related to the core subject of this dissertation.

6.5 Visual Approximate Policy Iteration

We now introduce a family $\mathcal{F}$ of state-action value function approximators that can be plugged in Nonparametric API to solve tasks that admit a raw feature generator, such as vision-for-action tasks. This family is inspired by the successful applications of the Extra-Trees that have just been discussed. Concretely, we take advantage of the following ideas:

- We use Extra-Tree models in the RL process, because they are powerful, yet fast and simple to implement. This idea comes from the Fitted Q Iteration algorithm [EGW05].

- As a raw feature generator can produce more than one raw feature per percept, an averaging technique is required. This is similar to image classification through Extra-Trees [MGPW05]. However, as we deal with supervised regression problems instead of supervised classification problems, the vote is replaced by a mean computation.

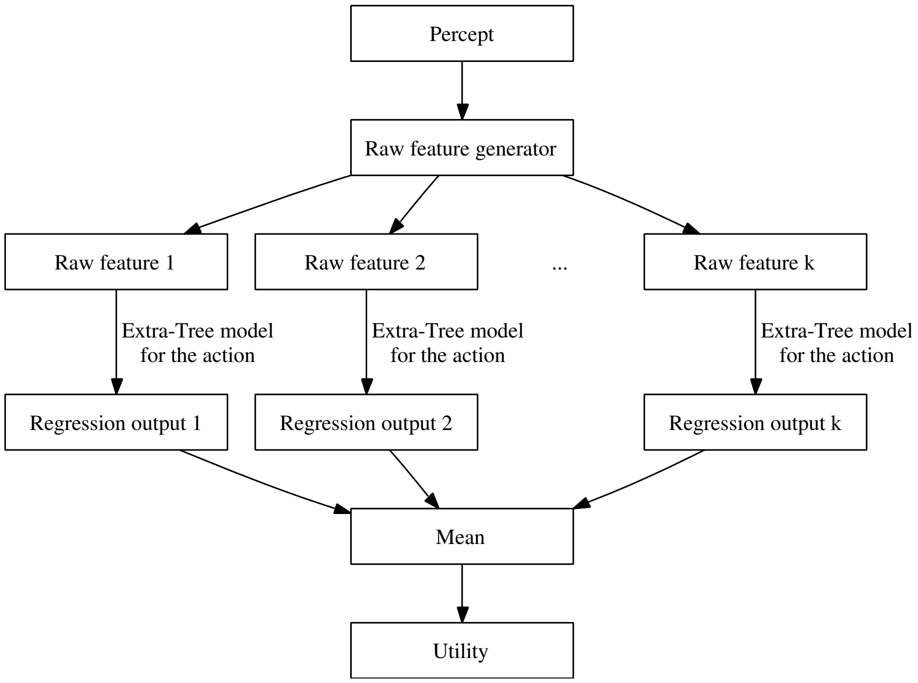


Figure 6.2: Computing the state-action utility of a raw percept for a fixed action. This architecture is derived from Figure 6.1.

A function approximator in this family is defined as a set of $|A|$ Regression Extra-Trees, one for each action. When evaluating the utility of a state-action pair (s, a), each raw feature $\boldsymbol{f}$ that is generated by the raw feature generator $\mathcal{G}_R$ for the percept s is fed to the Extra-Tree model that corresponds to the action a. Finally, all the outputs are averaged so as to obtain the utility of the percept-action pair. This process is illustrated in Figure 6.2. Formally, we have the following definition:

Definition 6.4. Let $\mathcal{G}_R$ be a raw feature generator. A *raw Extra-Tree state-action value function approximator* (or, for short, a RETQ approximator) is a tuple:

$$\hat{Q} = \left\langle \hat{Q}_{a_1}, \ldots, \hat{Q}_{a_m} \right\rangle , \tag{6.20}$$

where the finite set of actions is $A = \{a_1, \ldots, a_m\}$, and where each $\hat{Q}_a$ is a Regression Extra-Tree model that takes the raw feature space $\mathbb{R}^n$ as input. The output of a RETQ approximator for a state-action pair $(s, a) \in S \times A$ is defined as:

$$\hat{Q}(s, a) = \text{mean}\left(\left\{ \hat{Q}_a(\boldsymbol{f}) \mid \boldsymbol{f} \in \mathcal{G}_R(s) \right\} \right) . \tag{6.21}$$

The learning oracle $\mathcal{L}$ that can learn RETQ approximators from a database of triples $\langle s_t, a_t, v_t \rangle$ is now introduced (cf. Definition 6.3). Similarly to the application of Extra-Trees to image classification, this oracle generates an intermediate database that contains one utility for each raw feature in each percept s_t. This utility corresponds to the utility that was associated to the percept s_t that has triggered the raw feature. The corresponding code is depicted in Algorithm 6.6.

Algorithm 6.6 — Learning oracle for RETQ approximators

```
$\mathcal{L}(\{\langle s_t, a_t, v_t \rangle\}) :-$
    for $a \in A$ do
        $D[a] \leftarrow \{\}$
    end for
    for $t \leftarrow 1$ to $T$ do
        for $\boldsymbol{f} \in \mathcal{G}_R(s_t)$ do
            $D[a_t] \leftarrow D[a_t] \cup \{\langle \boldsymbol{f}, v_t \rangle\}$
        end for
    end for
    for $a \in A$ do
        $Q[a] \leftarrow$ extra-trees$(D[a])$
    end for
    return $\langle Q[a_1], \ldots, Q[a_m] \rangle$
```

The *Visual Approximate Policy Iteration* algorithm (V-API) is defined as the conjunction of Nonparametric API with RETQ approximators [JBP06]. V-API can cope with any task for which it is possible to define a raw feature generator, and notably vision-for-action tasks. This will be demonstrated in Section 6.7.

Very recently, related work has been proposed by Ernst et al. [EMW06]. In their approach, Extra-Trees are also used to solve vision-for-action tasks. However, there are two main differences with V-API: (1) They only consider global visual features (i.e. raw feature generators that only produce one feature per visual percept), that happen to be a raw portion of a grayscale image; and (2) they use the Fitted Q Iteration framework instead of Nonparametric API.

6.6 Distributed Implementation of Extra-Trees

V-API applies Regression Extra-Trees to large databases. As a consequence, even if the computational expense of Extra-Trees is small with respect to other machine learning algorithms, the complexity of visual spaces still prevents their direct use in V-API. We propose to reduce this computational expense by taking advantage of the extremely parallelizable nature of Algorithm 6.4: Each execution of Algorithm 6.5 is totally independent of other instances of the same algorithm, and each subtree can be computed in a separate computational task. In other words, the learning of Extra-Tree models can be formulated as a so-called *bag of tasks* [CBS$^+$03, dCS$^+$04], where tasks are independent and can be processed on separate computer nodes, which can greatly reduce the learning time. Once all the tasks are completed, it is straightforward to merge all the subtrees to generate the Extra-Tree model. As the application of a learned Extra-Tree model is very fast, only the induction of Extra-Trees will be distributed, and not the process of applying a model.

6.6.1 Building Extra-Trees in a Cluster of Computers

Our implementation of Regression Extra-Trees follows this principle. We assume the presence of a cluster of computers, in addition to the computer that runs the V-API algorithm. The latter computer will be refereed to as the *user's host*. Furthermore, we assume that the user's host can log into any host in the cluster through the Unix `ssh` (*secure shell*) command [SSH06]. This login must be possible without typing a password, which can be for example be done by using a `ssh-agent`.

Each application of Algorithm 6.5 in Algorithm 6.4 is called a *task*. Each task is run independently through `ssh` on one computer in the cluster, and is provided with the same database $\langle \boldsymbol{x_i}, y_i \rangle$ of input-output pairs. A simple task scheduler runs on the user's host so as to keep track of which machines in the cluster have been assigned a task. The scheduler has also the responsibility for detecting failures in the computations (e.g. reboot, lack of storage or temporary space, as well as network or hardware failures), which can be done thanks to the flexible options for invoking and monitoring a `ssh` session. Because of the random component of Extra-Tree induction, each instance of the program constructs a different subtree. All these subtrees are collected back through the pipe that is opened by `ssh`, and the user's host progressively inserts all these subtrees to form the resulting Extra-Tree model.

Whenever a subtree is returned and a subtree induction task is still pending, the scheduler assigns this pending task to the host that has finished its computation. This is of course a very primitive task scheduler: More sophisticated scheduling strategies (e.g. featuring preemption) could be used as well without distorting our approach. Furthermore, nothing would prevent the use of schedulers that would distribute bag-of-tasks computations inside a peer-to-peer grid of computers [FK04, Bd06], such as OurGrid [OG06], hereby benefiting from the computational power of a much larger number of hosts.

Even with a cluster of computers, the resulting speedup is already huge: If N homogeneous hosts are used, the computation time roughly equals $\lceil M/N \rceil T + U$, where M is the number of subtrees to generate (it is a parameter of Algorithm 6.4), T is the mean amount of time for building one single subtree, and U corresponds to the distribution time of the database among all the hosts. Therefore, we argue that taking advantage of the highly parallelizable nature of Extra-Trees [GEW06] (and, possibly, of Random Forests [Bre01]) could potentially have a great economical impact on industrial image processing or real-time video surveillance (cf. Section 6.4.2).

6.6.2 The Database Distribution Problem

Unfortunately, if no attention is paid, the transmission overhead U can quickly become a bottleneck. Indeed, the same large database has to be sent to N hosts by the central task manager, which causes augmentation of network congestion. Thus, a direct use of `ssh` to transmit the database would heavily reduce the interest of the distributed implementation.

One plausible solution is to rely on the use of UDP multicasting to distribute the database. However, due to the lack of flow control in UDP, slow hosts will

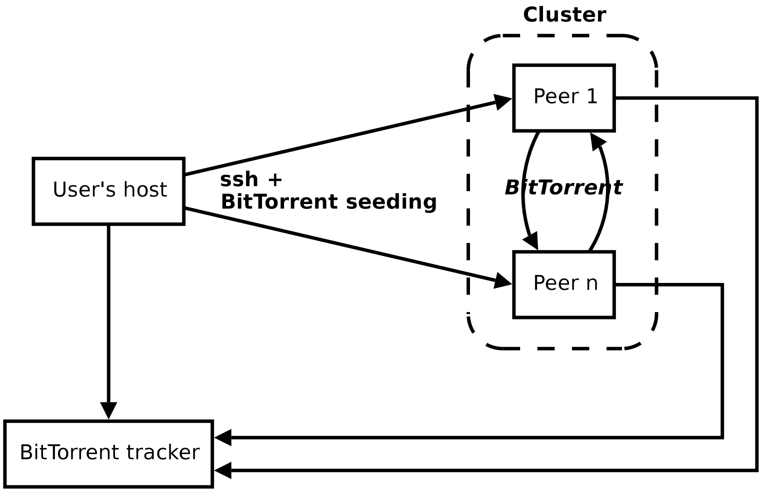


Figure 6.3: Software architecture for the distributed learning of Extra-Trees.

be overwhelmed by the massive amount of data they receive. As a consequence, a retransmission protocol would be required on top of UDP, which would greatly complicate the implementation. Furthermore, UDP multicasting is generally blocked by domain routers, which imposes the use of hosts inside the same domain. Another possibility is to design an *ad-hoc* data distribution protocol, that superimposes a virtual spanning tree topology on the hosts. The user's host, at the root of the spanning tree, would broadcast the databases to its children, and, in turn, each child in the topology would recursively repeat the received messages to its own children. This requires a still more complex implementation, because the spanning tree must be correctly maintained, even if new hosts are plugged into the cluster, or if a host is powered off or experiences difficulties. Moreover, experimental results show that the reliability layer of TCP itself acts as a bottleneck, because of the massive number of messages that are sequentially emitted by the host hierarchy.

We have therefore chosen to use the peer-to-peer protocol BitTorrent [Coh03, BT06] for distributing the databases. Schematically, in BitTorrent, each host interesting in downloading a file becomes part of a swarm that grabs all the pieces of the file. Whenever a host acquires a piece, this piece is made available for download to the other hosts. A distinguished host, the *tracker*, is used to keep track of the hosts that belong to the swarm [BTT06]. Our final learning architecture is schematized in Figure 6.3. Note that every time an host receives a database through BitTorrent, it caches it as a temporary file, so as to avoid a redundant file transfer if a new subtree induction task is thereafter scheduled on the same host.

This approach to file distribution is elegant and scalable, and indeed reduces the transmission overhead U, making it roughly independent of the number of hosts in the cluster. Such a solution for the distribution of databases over a network of computers is still vastly unexplored. It should be useful in many distributed computing applications in which all the tasks share the same input database, in particular in the context of Grid Computing [FK04, Bd06].

6.7 Experimental Results

Visual Approximate Policy Iteration was evaluated on the same benchmark visual task as used in Section 5.1.5. The experimental setup was kept exactly identical. Similarly to RLVC, V-API was applied to a static database of 10,000 interactions that were collected using a fully randomized exploration policy. Again, the same database is used throughout the entire V-API algorithm, and this database only contains images that belong to the learning set.

The parallel implementation of Regression Extra-Trees runs on a testbed cluster of $N = 67$ heterogeneous ix86 machines. It consists of a set of 27 AMD Athlon 1800+, 27 Intel Celeron 2.4GHz, and 13 Intel Pentium IV 2.8GHz CPUs. They are interconnected via a switched 100Mbps Ethernet network.

Figure 6.4 shows the sequence of policies that is generated by V-API. The algorithm stops after 6 iterations, which is a surprisingly small number. A total number of 49 RETQ approximators were generated, which corresponds to $49 \times |A| = 147$ Regression Extra-Tree models. The overall running time was about 98 hours. This shows the interest of taking advantage of the intrinsic parallelism of Extra-Trees, as this divided the running time by about fifty. Statistics about the generated policies are given in Figure 6.5. The error of the last policy in the generated sequence was 1.6% on the learning set and 6.6% on the test set, with respect to the optimal policy when the agent has direct access to its position and viewing direction. These error rates are better than those obtained in Section 5.1.5 through the basic version of RLVC (0.1% on the learning and 8% on the test set), but they are slightly inferior to that of RLVC extended with classifier compaction (0% on the learning and 4.5% on the test set). However, the better balance between errors on the learning set and on the test set indicates that overfitting effects are less present. Note also that these error rates are notably a consequence of the downscaling of the input images to a size of 160 × 120 that was used to reduce the abundance of detected visual features.

The advantage of using BitTorrent is clear: Under optimal conditions, if m Extra-Tree models (as discussed above, $m = 147$) are to be built from databases of typical size $S = 80$MB, the transmission overhead corresponding to FTP would have a theoretical lower bound of $U \approx m \times N \times S[\text{MB}]/100[\text{Mbps}] = 17$ hours. Using BitTorrent, this time is approximately reduced to $U \approx m \times S[\text{MB}]/100[\text{Mbps}] = 16$ minutes.

6.8 Summary

We have defined the Visual Approximate Policy Iteration (V-API) algorithm that is an instance of API designed to work notably in visual spaces. It is based on a non-parametric version of API. V-API extensively relies on the use of Regression Extra-Trees as function approximators. This choice is motivated by the low bias and variance, as well as by the good generalization performance of the Extra-Trees [GEW06]. Furthermore, Classification Extra-Trees have been proved to be successful at solving image classification tasks [MGPW05]. The embedding of Extra-Trees inside the

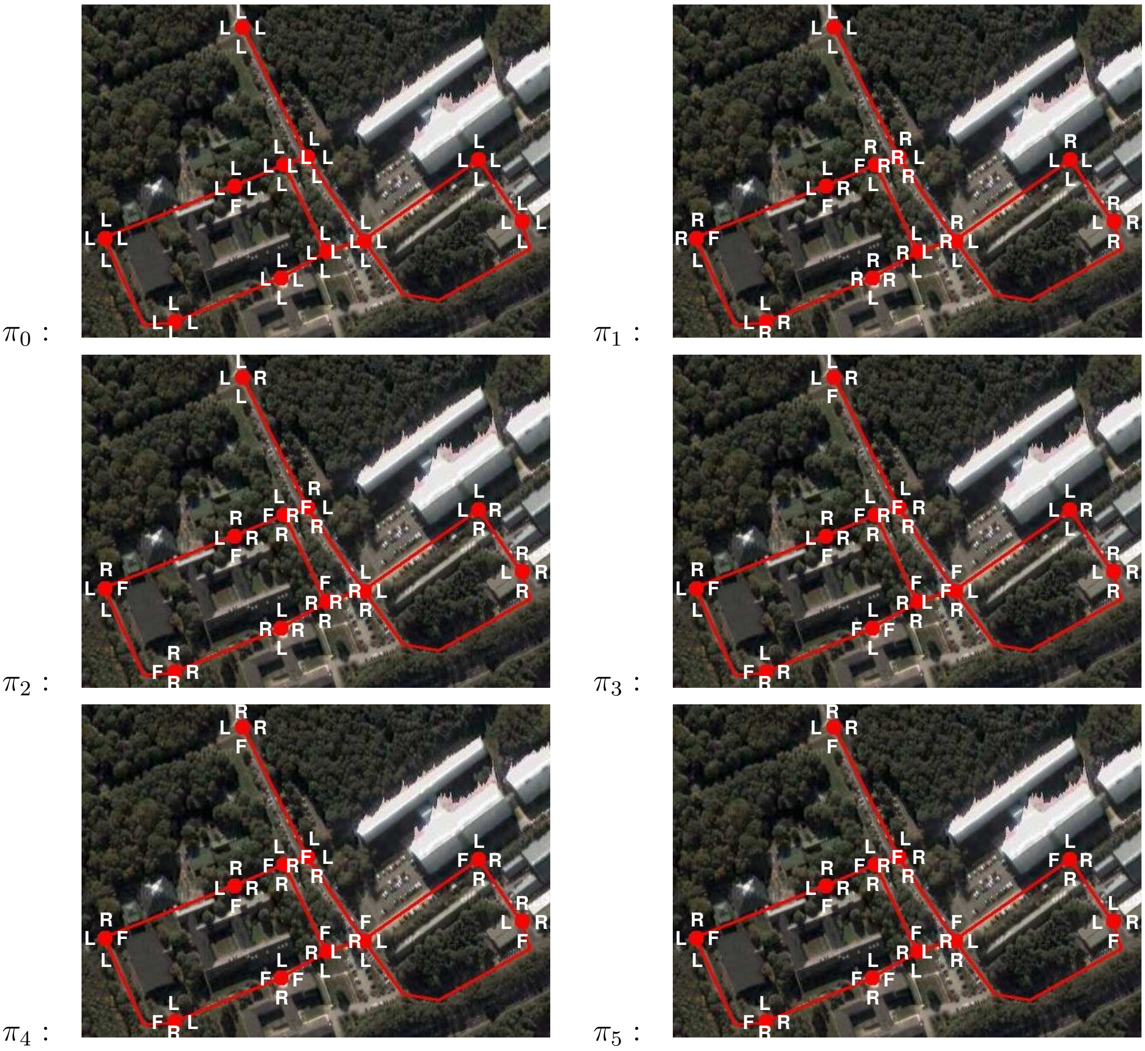


Figure 6.4: The sequence of policies π_k that are generated by V-API. At each location, 4 letters from the set {F, L, R} are written, one for each viewing direction. Each letter represents the action (go Forward, turn Left, turn Right) that receives most votes in the learning set, for the corresponding pair *location / viewing direction*.

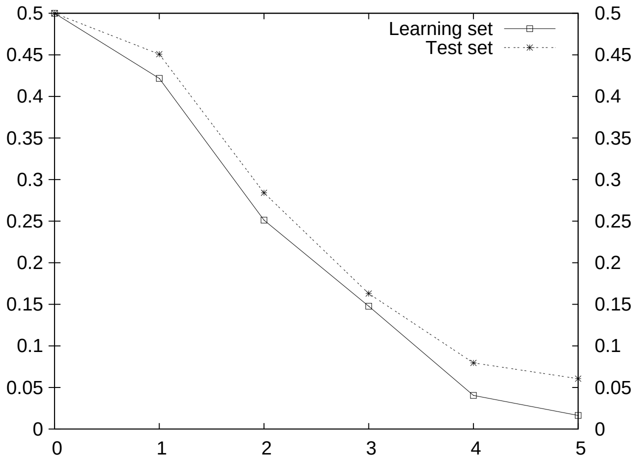


Figure 6.5: Error on the generated image-to-action mappings as a function of the step counter k. The solid (resp. dashed) plot corresponds to the error rate of the policy π_k on the learning (resp. test) set.

framework of API is a first important contribution of this chapter, and should also be useful in continuous perceptual spaces. To the best of our knowledge, this makes of V-API the first application of API to high-dimensional, *discrete* spaces.

We have also shown how to take advantage of the highly parallelizable nature of the induction of Extra-Trees by distributing the construction of the subtrees over a cluster of computers. This allows us to greatly reduce the computational time, which is often an important issue with visual spaces. Finally, the peer-to-peer BitTorrent protocol was shown to be an effective tool for reducing the database distribution expense. Of course, other fields of application of Extra-Trees, such as supervised learning [GEW06], image classification [MGPW05] and the Fitted Q Iteration algorithm [EGW05], will directly benefit from this new economical advantage.

Experimental results show similar results between RLVC and V-API. Here are the main advantages of V-API over RLVC:

- It is conceptually simple and elegant;
- There is no need to cluster the raw features (e.g. by introducing a metric), thus it benefits from the full discriminative power of the raw features;
- It exhibits the low variance and bias of Extra-Trees, so that the robustness and the capabilities of generalization are potentially better, and so that it is more stable.

However, these competitive advantages of V-API must be balanced by the following disadvantages with respect to RLVC:

- The computational expense is much higher;
- The use of a cluster of computers is almost mandatory, which heavily complicates the implementation;

- It is impossible to generate higher-level features on demand (cf. Chapter 5).

Summarizing, a compromise seems to exist between the requirement of an equivalence relation among visual features, and the use of raw visual features. Therefore, V-API and RLVC certainly constitute two complementary techniques.

Acknowledgments

We reproduce here the acknowledgments that figure in the seminal paper about Visual Approximate Policy Iteration [JBP06]. We kindly acknowledge P. Geurts and the PEPITe S.A. team (http://www.pepite.be/) for providing us with an implementation of Extra-Trees, which has inspired our novel parallel implementation of Extra-Trees. We also wish to thank S. Martin of the University of Liège for his valuable help regarding network programming and the protocol BitTorrent.

CHAPTER

SEVEN

Reinforcement Learning of Joint Classes

Paradoxically, although robotic controllers often interact with their environment through a set of continuously-valued actions (position, velocity, torque...), relatively little consideration has been given to the development of RL algorithms that would learn direct mappings from percepts to continuous *actions. This is in contrast to continuous perceptual spaces, for which many solutions exist (cf. Section 6.2). As a consequence, vision-based robotic tasks with continuous output such as visual servoing cannot currently be solved through RL. Moreover, even if the action space is finite, standard reinforcement learning algorithms are doomed to failure if there is a large number of possible actions. This is mainly due to the use of lookup tables to represent the value functions, which makes the database of interactions dispersed over a large number of cells.*

The challenge posed by continuous actions spaces more fundamentally arises from the fact that standard update rules based upon Bellman backup operators are only applicable to finite sets of actions, as these rely on a maximization over the action space. Furthermore, an a priori *discretization of the action space generally suffers from an explosion of the representational size of the domains known as the* curse of dimensionality*, and may introduce artificial noise. As a consequence, similarly to adaptive-resolutions techniques for perceptual spaces (cf. Section 4.3.3* and *4.3.4), it seems promising to introduce adaptive-resolution methods for* action spaces.

This chapter presents the Reinforcement Learning of Joint Classes *(RLJC) algorithm, which enables such closed-loop learning of direct mappings from images to complex action spaces. RLJC is a generalization of RLVC to continuous or high-dimensional action spaces. Rather than focusing the attention of the agent only on highly informative features in the perceptual space, RLJC also takes features over the action space into consideration. Therefore, RLJC is an adaptive-resolution method for the* joint space of percepts and actions.

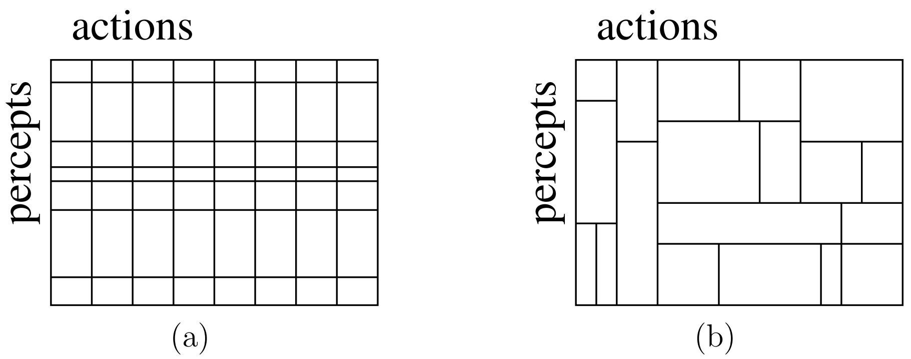


Figure 7.1: Illustration of the discretization process of (a) RLVC, and (b) RLJC.

7.1 Adaptive Resolution in the Joint Space

As proposed above, a natural idea for extending RLVC to continuous action spaces is to discretize the action space together with the perceptual space. *Reinforcement Learning of Joint Classes* (RLJC) follows this principle, and subdivides the problem space in a finite set of regions by applying tests to both the input perceptual space *and* the output action space. More precisely, RLJC tests the presence of *perceptual features* and of *action features*, that are both highly informative for the task to be solved. For instance, if uni-dimensional continuous actions are considered, an action feature could be a real number that would serve as a threshold on the effector output, and that would mark off a region that has specific properties with respect to the current task.

Similarly to RLVC that incrementally subdivides the perceptual space S, RLJC progressively subdivides the combined percept-action space (or *joint* space) $S \times A$, in a sequence of attempts to remove perceptual aliasing. In each region that is induced by the discretization process, the state-action value functions are kept constant. This way, the possibly uncountable joint space is mapped to a finite number of regions, and specific RL algorithms are then used to extract the optimal control policies. Very importantly, the discretization process is *adaptive*: A new split occurs only when it succeeds at distinguishing between two regions of the joint space that have dissimilar properties with respect to the current state-action value function. Therefore, the discretization of the action space can be inhomogeneous with respect to the perceptual space. In other words, the action space can possibly be discretized differently at each percept. This difference is illustrated in Figure 7.1.

7.2 Related Work

Previously-investigated solutions for handling continuous actions without *a priori* discretization generally use function approximators such as neural networks [GSK98, tHK00] or tile coding [SSR98]. Early work on neural networks in continuous action space is reviewed by Kaelbling et al. [KLM96, Section 6.2]. Rummery proposes several methods based on a collaboration between Adaptive Heuristic Critic (cf. Section 2.4.3) and neural networks [Rum95, Chapter 5]. Ernst et al. mention that Fitted Q Iteration together with Extra-Trees (cf. Section 6.4.2) could be used in continuous action spaces [EGW05, Section 4.5], though an experimental validation is not given.

However, such function approximators are not well suited to continuous action spaces. Indeed, although they can be used in approximate actor-critic architectures such AHC, they are not directly applicable to algorithms such as Q-learning, SARSA, Fitted Value Iteration, or Approximate Policy Iteration. This is a consequence of the fact that these families of approximators cannot directly find the action that maximizes the state-action value function given an input percept. Mathematically, this operation corresponds to a multi-dimensional optimization of the projection of a state-action value approximator over the action space. To this end, iterative techniques for optimization of non-linear, non-convex, noisy functions are in general required. This means that potentially a large number of actions have to be evaluated before a good estimate of the optimal action is obtained. As the evaluation of the utility of a state-action pair can be computationally expensive, this process might be inapplicable in practice. Moreover, state-of-the-art optimization techniques such as the *simplex algorithm* for general function maximization [NM65] are likely to be trapped in local optima. Although they have been designed to avoid such a deficiency, there is no guarantee that algorithms such as the *stochastic hill climbing* method or the *genetic algorithms* [Hol75] will either reach a global optimum. Nevertheless, success is reported by Smart [Sma02, Section 4.8] on uni-dimensional actions by using pairwise bisection [AMC00].

From this point of view, *wire fitting* is a particularly attractive family of function approximators, because it supports the quick and exact extraction of the optimal action [BK93] for a given percept. The underlying methodology is a moving least-squares interpolator. The original idea of wire fitting was later extended by Gaskett et al. using artificial neural networks, leading to the *wire fitted neural networks* that have been exploited in the Q-learning framework [GWZ99, Gas02]. Nevertheless, wire fitting is inherently defined over continuous joint spaces, which forbids its use for solving visual tasks.

On the other hand, all the adaptive-resolution methods that were discussed in Section 4.3 assume a finite number of actions. Moreover, in practice, these actions cannot be too numerous. A noticeable exception is the work by Porta and Celaya [PC05], who consider feature detectors that are defined over the action space. Unfortunately, this approach assumes that the set of relevant action features is selected by hand. This contrasts with RLJC that is fully automatic, and that can draw features from a possibly infinite set of features. The basic idea of RLJC (i.e. discretizing the joint percept-action space) is also present in the *JoSTLe* algorithm [MWSP04]. JoSTLe is one of the few adaptive-resolution techniques that can discretize a continuous action space. It is an extension of Variable Resolution Grids [MM02]. However, JoSTLe is specifically designed for fully continuous control problems, as it heavily relies on Kuhn triangulations of the joint space.

Summarizing, to the best of our knowledge, none of these methods has been designed to cope with high-dimensional, *discrete* perceptual spaces such as visual spaces, simultaneously with continuous action spaces. In contrast, RLJC is not limited either to discrete or to continuous joint spaces: Indeed, RLJC only requires that features can be defined on the perceptual and on the action spaces. Therefore, one key advantage of RLJC lies in its generality.

7.3 Joint Features

7.3.1 Features in the Action Space

Just like perceptual features can be defined over the perceptual space (cf. Section 4.1.2), it possible to define *action features* over the action space. An action feature is anything that can be either present or absent from one of the agent's effector outputs. Just like a perceptual feature, an action feature splits the action space in two parts. This leads us to the following definitions:

Definition 7.1. An *action feature detector* $\mathcal{D}_A : A \times F_A \mapsto \mathcal{B}$ is a mapping that associates a Boolean to any pair that consists of an action and of an action feature[1]. F_A is the *action feature space*, that is possibly infinite.

Besides the action feature detector, the presence of an oracle that extracts action features from effector outputs is assumed. This generator computes the action features that a given set of actions exhibits. The corresponding definition can be naturally derived from the notion of a perceptual feature generator (cf. Section 4.1.3):

Definition 7.2. An *action feature generator* $\mathcal{G}_A : \mathcal{P}(A) \mapsto \mathcal{P}(F_A)$ is a mapping that associates to each set of actions, the set of all the action features that are present in one of these actions. For computational tractability, it is required that for each finite set $\{a_1, \ldots, a_m\}$ of actions, $\mathcal{G}_S(\{a_1, \ldots, a_m\})$ is finite.

7.3.2 Features for Continuous Action Spaces

We are primarily interested in closed-loop learning of mappings from images to continuous actions. In this case, $A = \mathbb{R}^n$ for some positive integer n. Suitable action features are mostly identical to the features for continuous perceptual spaces (cf. Section 4.1.2). The corresponding action features simply consist in testing a threshold on a particular component of the action space. The set of action features is defined as:

$$F_A = \mathbb{R} \times \{1, \ldots, n\}. \tag{7.1}$$

The corresponding action feature detector $\mathcal{D}_A$ checks whether the considered component of the given action is below the threshold or not:

$$\mathcal{D}_A(\boldsymbol{a}, (t, i)) = \textbf{true} \text{ if and only if } a_i < t. \tag{7.2}$$

[1] Similarly to Definition 4.4, the subscript A of the notation $\mathcal{D}_A$ and F_A emphasizes the fact that these features are defined on the action space A.

Finally, the action feature generator $\mathcal{G}_A$ converts any action to n action features, one for each component of the input action $\boldsymbol{a} \in \mathbb{R}^n$:

$$\mathcal{G}_A(\{\boldsymbol{a_1}, \dots, \boldsymbol{a_m}\}) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n} (a_{ij}, j). \tag{7.3}$$

7.3.3 Features for Cartesian Action Spaces

As discussed in Section 7.2, Porta and Celaya consider high-dimensional, discrete action spaces [PC05]. In their framework, an action governs a set of k actuators. The output value for the ith actuator must lie in a finite domain: $A_i = \{1, \dots, n_i\}$. Therefore, the action space is the Cartesian product of k finite domains:

$$A = \bigtimes_{i=1}^{k} \{1, \dots, n_i\}. \tag{7.4}$$

Such a Cartesian action space is basically the direct transposition of the binary-number perceptual features (cf. Section 4.1.2) to action spaces and to variables with more than two possible values. In this case, a perceptual feature tests whether the ith component of an action is labeled by a given value:

$$\begin{aligned} F_A &= \bigcup_{i=1}^{k} \{(t, j) \mid j \in \{1, \dots, n_i\}\}, && (7.5) \\ \mathcal{D}_A(\boldsymbol{a}, (t, i)) &= \textbf{true} \text{ if and only if } a_i = t. && (7.6) \end{aligned}$$

As the set of action features is finite, the action feature generator can simply return the whole set of possible action features:

$$\mathcal{G}_A(\{a_1, \dots, a_m\}) = F_A. \tag{7.7}$$

A better strategy that reduces the number of features consists in returning only the features that are present in the set of actions:

$$\mathcal{G}_A(\{a_1, \dots, a_m\}) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n_i} (a_{ij}, j). \tag{7.8}$$

7.3.4 Joint Feature Detectors and Generators

When features are defined both over the perceptual space S and over the action space A, it is possible to define *joint features*, that are either perceptual features or action features:

Definition 7.3. Let F_S (resp. F_A) be a set of perceptual (resp. action) features. The *joint feature space* is defined as:

$$F = F_S \cup F_A. \tag{7.9}$$

Then, the feature detectors and generators can be trivially extended to the joint space as follows:

Definition 7.4. Let $\mathcal{D}_S$ (resp. $\mathcal{D}_A$) be a perceptual (resp. action) feature detector. The *joint feature detector* $\mathcal{D} : (S \times A) \times F \mapsto \mathcal{B}$ that is induced by $\mathcal{D}_S$ and $\mathcal{D}_A$ is a mapping that associates a Boolean to any triple that consists of a state-action pair and of a joint feature. Such a detector is defined through the relation:

$$\mathcal{D}((s,a),f) = \begin{cases} \mathcal{D}_S(s,f) & \text{if } f \in F_S, \\ \mathcal{D}_A(a,f) & \text{otherwise.} \end{cases} \tag{7.10}$$

Definition 7.5. Let $\mathcal{G}_S$ (resp. $\mathcal{G}_A$) be a perceptual (resp. action) feature generator. The *joint feature generator* $\mathcal{G} : \mathcal{P}(S \times A) \mapsto \mathcal{P}(F)$ that is induced by $\mathcal{G}_S$ and $\mathcal{G}_A$ is a mapping that associates to each set of state-action pair, the set of all the joint features that are present in one of these state-action pairs:

$$\mathcal{G}(\{(s_1,a_1),\ldots,(s_m,a_m)\}) = \mathcal{G}_S(\{s_1,\ldots,s_m\}) \cup \mathcal{G}_A(\{a_1,\ldots,a_m\}). \tag{7.11}$$

These concepts will prove useful in the next sections, in which RLJC is formally developed.

7.4 Reinforcement Learning of Joint Classes

Generalizing RLVC to continuous action spaces raises two important difficulties. Firstly, because the discretization of the action space is non-uniform with respect to the perceptual space, standard reinforcement learning algorithms for finite state-action spaces (cf. Chapter 2) are useless. This contrasts with RLVC, that can take advantage of such algorithms by working on the mapped MDP $\mathcal{M}_k$ (cf. Section 4.4.1). Secondly, the aliasing criterion defined in Section 4.4.2 cannot be used directly when the action space is continuous, for at least two reasons: The Bellman residuals as defined by Equation 4.23 are unavailable because of the non-uniform discretization of the action space; and the set of time stamps of Equation 4.24 is useless, because the action space can only be sparsely sampled, so that any $T(c,a)$ essentially collapses to a set containing at most one element. We now describe the structure of RLJC, then how these difficulties are overcome.

7.4.1 Learning Architecture

Whereas RLVC learns a sequence of *percept classifiers* $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots$ that discretize the percept space by testing perceptual features, RLJC learns a sequence of *joint classifiers* $\mathcal{J}_0, \mathcal{J}_1, \mathcal{J}_2, \ldots$ that discretize the joint state-action space by testing features on the perceptual *and* on the action space. In terms of the notation of Section 7.3, a joint classifier $\mathcal{J}_k$ in this sequence is a binary decision tree, the internal nodes of which are labeled by a joint feature. Each leaf of these decision trees defines a *joint class*. To classify a state-action pair $(s,a) \in S \times A$, the system starts at the root node, then progresses down the tree according to the result of the joint feature detector $\mathcal{D}$ for each joint feature found during the descent, until reaching a leaf. Thus, a joint classifier maps the (possibly infinite) joint space $S \times A$ to a finite set of joint classes using the joint feature detector $\mathcal{D}$:

Definition 7.6. A *joint classifier* $\mathcal{J}_k : S \times A \mapsto C_k$ is a mapping from the joint space to a finite set of m_k joint classes C_k that is defined as:

$$C_k = \left\{ c_1^{(k)}, \ldots, c_{m_k}^{(k)} \right\}. \tag{7.12}$$

In RLVC, for any state-action value function, all the percepts $s, s' \in c$ that lie in the same perceptual class c share the same utility for all the actions. Similarly, once a joint classifier $\mathcal{J}_k$ is fixed in RLJC, it constrains the allowed structure of the state-action value functions $Q(s, a)$, by requiring them to be piecewise constant over each joint class:

Definition 7.7. Let $\mathcal{J}_k$ be a perceptual joint class. A *constrained interpretation* $\hat{Q} : C_k \mapsto \mathbb{R}$ is a mapping that associates a real-valued utility to each joint class. In turn, each constrained interpretation induces a *constrained state-action value function* $Q : S \times A \mapsto \mathbb{R}$ as follows:

$$Q(s, a) = \hat{Q}(\mathcal{J}_k(s, a)), \tag{7.13}$$

where $s \in S$ is a percept and $a \in A$ is an action.

The general scheme of RLJC is identical to that of RLVC. The algorithm starts with a joint classifier $\mathcal{J}_0$ that contains one single leaf, and thus that maps all of its input state-action pairs to a single perceptual class $c_1^{(0)}$. For each $\mathcal{J}_k$ in the sequence, the optimal state-action value function Q_k^* constrained by $\mathcal{J}_k$ is computed. Then, a set of highly discriminative joint features are selected by relying on an analysis of the Bellman residuals that are induced by Q_k^*. The selected features are used to refine $\mathcal{J}_k$, leading to the joint classifier $\mathcal{J}_{k+1}$. New joint classifiers are generated until perceptual aliasing vanishes. The resulting backbone of RLJC is formalized in Algorithm 7.1, and is visibly very similar to Algorithm 4.1. However, several additional difficulties are hidden in this algorithm:

1. Given a state-action value function that is constrained by a joint classifier $\mathcal{J}_k$, how is it possible to extract the greedy action with respect to a percept $s \in S$? This operation is indeed required by the reinforcement learning algorithm and when controlling the system after RLJC has completed. The non-uniform discretization complicates this operation, that was trivial in the context of RLVC.

2. Given a joint classifier $\mathcal{J}_k$, how is it possible to compute the optimal state-action value function Q_k^* that is constrained by $\mathcal{J}_k$?

3. How to adapt the aliasing detector and generator that have been defined in the framework of RLVC in Chapter 4?

These problems are discussed in the next sections, which will lead to a concrete implementation of Algorithm 7.1.

Algorithm 7.1 — General structure of RLJC

```
$k \leftarrow 0$
$m_k \leftarrow 1$
$\mathcal{J}_k \leftarrow$ binary decision tree with one leaf
repeat
    Collect $N$ interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$
    Reinforcement learning of $Q^*_k(s,a)$, as constrained by $\mathcal{J}_k$
    $m_{k+1} \leftarrow m_k$
    $\mathcal{J}_{k+1} \leftarrow \mathcal{J}_k$
    for $i \leftarrow 1$ to $m_k$ do
        $c \leftarrow c_i^{(k)}$
        if aliased$(c)$ then
            $f^* \leftarrow$ selector$(c)$
            if $f^* \neq \perp$ then
                $\mathcal{J}_{k+1} \leftarrow \mathcal{J}_{k+1}$, where the joint class $c$ is refined by a test on $f^*$
                $m_{k+1} \leftarrow m_{k+1} + 1$
            end if
        end if
    end for
    $k \leftarrow k + 1$
until $\mathcal{J}_k = \mathcal{J}_{k-1}$
```

Algorithm 7.2 — Computing the utility of a percept $s \in S$ in RLJC

```
utility$(s, Q, \mathcal{J}_k)$ :−
    $C \leftarrow \{c \mid (\exists a \in A)\ \mathcal{J}_k(s,a) = c\}$
    return $\max_{c \in C} \hat{Q}(c)$
```

7.4.2 Computing a Greedy Action

In this section, we describe how to effectively compute the greedy action

$$\pi[Q]\,(s) = \operatorname*{argsup}_{a \in A} Q(s,a), \tag{7.14}$$

for a percept $s \in S$, where Q is a state-action value function that is constrained by some joint classifier $\mathcal{J}_k$, whose constrained interpretation is $\hat{Q}$. Thanks to the parametrization of constrained state-action value functions, it is possible to extract $\pi[Q]\,(s)$ by evaluating $Q(s,a)$ only for a finite number of actions.

Compatible Joint Classes

To this end, we first evaluate the set of joint classes that are *compatible* with the percept s:

$$C(s) = \{c \mid (\exists a \in A)\ \mathcal{J}_k(s,a) = c\}\,. \tag{7.15}$$

Intuitively, this set corresponds the joint classes to which a given percept may be mapped. This is illustrated in Figure 7.2. From an algorithmic point of view, a joint

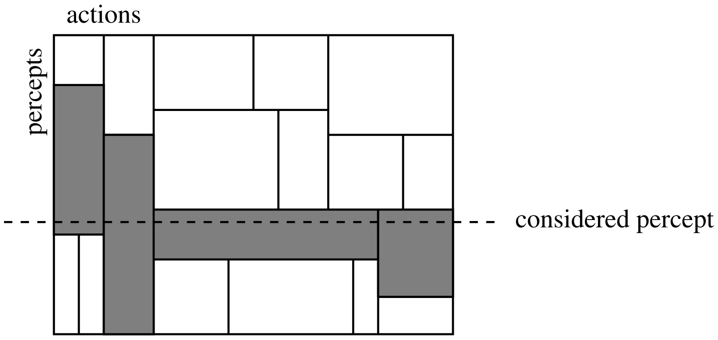


Figure 7.2: The joint classes that are compatible with a given percept through a joint classifier. These classes are grayed in this illustration. To compute the utility of the considered percept, it is sufficient to maximize the constrained interpretation over these compatible classes.

class c is compatible with a percept if it is possible, starting from the leaf of $\mathcal{J}_k$ that corresponds to c, to climb the decision tree up to the root node without violating any test on a perceptual feature. This gives an effective procedure for evaluating $C(s)$ through a top-down exploration: For each path from the root node to a leaf, the corresponding leaf is added if and only if the percept s violates none of the tests on perceptual features that label this path.

Evaluating the Utility of a Percept

The set of compatible classes is finite, as the set of joint classes is itself finite. Let us now call $c^* \in C_k$, the *optimal compatible* joint class, to wit, the class that maximizes the constrained interpretation $\hat{Q}$ among the joint classes that are compatible with s:

$$c^* = \operatorname*{argmax}_{c \in C(s)} \hat{Q}(c). \tag{7.16}$$

Then, using Equation 7.13, the utility of the percept s, that corresponds to the utility of the greedy action $\pi[Q]\,(s)$, can be evaluated as follows:

$$\sup_{a \in A} Q(s,a) = \sup_{a \in A} \hat{Q}(\mathcal{J}_k(s,a)) = \hat{Q}(c^*). \tag{7.17}$$

This equation means that the utility of a percept s corresponds to the state-action value that is maximal among the joint classes that are compatible with s. Algorithm 7.2 summarizes this discussion.

Constructing a Greedy Action

Now, a greedy action $\pi[Q]\,(s)$ can be implicitly defined as one of the actions that allows to travel from the root node to the optimal compatible joint class c^*:

$$\mathcal{J}_k(s, \pi[Q]\,(s)) = c^*. \tag{7.18}$$

Algorithm 7.3 — Computing the greedy action $\pi[Q](s)$ for some percept $s \in S$

```
greedy-action $(s, Q, \mathcal{J}_k)$ :–
    $C \leftarrow \{c \mid (\exists a \in A)\ \mathcal{J}_k(s, a) = c\}$
    $c^* \leftarrow \operatorname{argmax}_{c \in C} \hat{Q}(c)$
    $\Lambda \leftarrow$ path in the tree $\mathcal{J}_k$ that leads from the root node to the leaf $c^*$
    $F_\oplus \leftarrow$ set of sustained action features along $\Lambda$
    $F_\ominus \leftarrow$ set of negated action features along $\Lambda$
    return action-seeker$(F_\oplus, F_\ominus)$
```

To turn this process into a working algorithm, an additional component is needed:

Definition 7.8. An *action seeker* $\mathcal{A} : \mathcal{P}(F_A) \times \mathcal{P}(F_A) \mapsto A$ is an oracle that, given a finite set of *required* action features $F_\oplus \subset F_A$ and a finite set of *forbidden* action features $F_\ominus \subset F_A$, returns an action $a \in A$ that exhibits all the required action features, and that exhibits none of the forbidden action features:

$$\mathcal{D}_A(\mathcal{A}(F_\oplus, F_\ominus), f) = \textbf{true} \quad \text{for all } f \in F_\oplus, \tag{7.19}$$

$$\mathcal{D}_A(\mathcal{A}(F_\oplus, F_\ominus), f) = \textbf{false} \quad \text{for all } f \in F_\ominus. \tag{7.20}$$

Thanks to the action seeker, it is possible to compute a greedy action $\pi[Q](s)$ by asking the action seeker to find an action that fulfills all the tests on the action features that label the path from the root node to the optimal class c^*. This path can be extracted through a bottom-up walk in the tree, starting from the leaf that corresponds to c^*. This walk can be implemented efficiently, as long as each node of the tree is associated with a pointer to its parent node. The corresponding piece of code is summarized in Algorithm 7.3.

Example 7.9. The principle behind Algorithm 7.3 is illustrated in Figure 7.3. In this example, the perceptual features along the path Λ are $\{\mathit{fs1}, \mathit{fs2}, \mathit{fs3}\} \subset F_S$, and the action features are $\{\mathit{fa1}, \mathit{fa2}, \mathit{fa3}\} \subset F_A$. Let us assume that we are given a percept $s \in S$ that violates none of the tests on perceptual features along the path Λ: In other words, s must exhibit *fs2*, but neither *fs1*, nor *fs3*. Now, the greedy action $\pi[Q](s)$ must bring $\mathcal{J}_k$ to c^* when s is given as input. This implies that this greedy action must sustain the action feature *fa1*, while negating action features *fa2* and *fa3*. Thus, $F_\oplus = \{\mathit{fa1}\}$ and $F_\ominus = \{\mathit{fa2}, \mathit{fa3}\}$ in this case. Finally, the action seeker can be used to extract an action $a \in A$ that fulfills the features in $F_\oplus$ while the features in $F_\ominus$ are negated. The action a is greedy by construction, as it brings the state-action pair (s, a) to the optimal compatible joint class c^*.

Action Seeker for Continuous Actions

We now define an action seeker that is suited to continuous action features (cf. Section 7.3.2). In this context, the required set $F_\oplus$ of features along with the forbidden set $F_\ominus$ of features defines the following region R of the action space $A = \mathbb{R}^n$:

$$R = \{\boldsymbol{a} \in \mathbb{R}^n \mid ((\forall (t, i) \in F_\oplus)\ a_i < t) \text{ and } ((\forall (t, i) \in F_\ominus)\ a_i \geq t)\}. \tag{7.21}$$

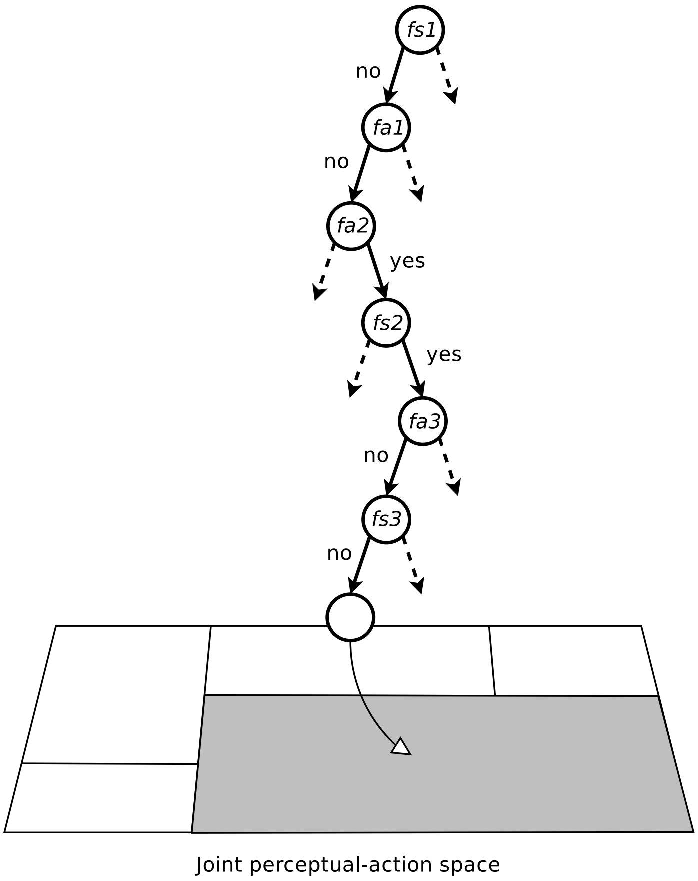


Figure 7.3: A path Λ from the root node to an optimal compatible joint class c^* in a hypothetical joint classifier.

Clearly, this region R corresponds to a hyperrectangle, whose lower bounds are constrained by the forbidden features, whereas its upper bounds are given by the required features. A natural action seeker therefore consists in returning a point inside the hyperrectangle, for example the center of the hyperrectangle. This process is formalized in Algorithm 7.4. In practice, A is itself a bounded hyperrectangle, because each output channel has an allowed range. Thus, Algorithm 7.4 takes these ranges into consideration: b_i (resp. c_i) corresponds to the lower (resp. upper) bound of the allowed range for the ith effector channel.

7.4.3 Reinforcement Learning through Joint Classifiers

The second key problem in RLJC is to be able to obtain an optimal state-action value function that is constrained by a joint classifier $\mathcal{J}_k$. For the purpose of this discussion, we first draw some observations about the relations between constrained state-action value functions and function approximation methods in reinforcement learning.

Algorithm 7.4 — Action seeker for continuous action spaces

```
action-seeker ($F_\oplus$, $F_\ominus$) :–
    for $(t, i) \in F_\ominus$ do
        if $b[i] < t$ then
            $b[i] \leftarrow t$
        end if
    end for
    for $(t, i) \in F_\oplus$ do
        if $c[i] > t$ then
            $c[i] \leftarrow t$
        end if
    end for
    for $i \leftarrow 1$ to $n$ do
        $a_i \leftarrow (b[i] + c[i])/2$
    end for
    return $\boldsymbol{a}$
```

Relating RLVC to Function Approximation

Although it has not been written explicitly, all the adaptive-resolution techniques from Sections 4.3.3 and 4.3.4 can be thought of as specialized function approximation methods. This notably means that the learning of the optimal state-action value function for the MDP that is mapped through a percept classifier $\mathcal{C}_k$ (cf. Section 4.4.1) is inherently a function approximation process.

Indeed, once a percept classifier $\mathcal{C}_k$ is fixed in RLVC, all the state-action value functions that are computed are piecewise constant in each perceptual class with respect to each action. More precisely, let $Q_k(s, a)$ be a state-action value function that is constrained by $\mathcal{C}_k$. Then, all the percepts $s, s' \in c$ that lie in the same perceptual class c are required to share the same value:

$$Q_k(s, a) = Q_k(s', a), \tag{7.22}$$

for all $a \in A$. As a consequence, these functions can be parametrized by a matrix W of real numbers of dimensions $|S_k| \times |A|$:

$$Q_k(s, a; W) = \sum_{i=1}^{|S_k|} \sum_{j=1}^{|A|} W_{ij} \phi_{ij}(s, a), \tag{7.23}$$

where $\phi_{ij}(s, a)$ is a function that indicates whether the percept s is mapped in the ith perceptual class, while action a simultaneously corresponds to the jth possible action:

$$\phi_{ij}(s, a) = \begin{cases} 1 & \text{if } \mathcal{C}_k(s) = c \text{ and } a = a_j, \\ 0 & \text{otherwise,} \end{cases} \tag{7.24}$$

if the finite set of actions is $A = \{a_1, \ldots, a_m\}$. As a consequence, the process of computing the optimal state-action value function $Q_k^*(s, a)$ of a mapped MDP is equivalent to the process of computing optimal parameters W^* for the linear approximation architecture that is described by Equation 7.23. Such approximation architectures were discussed in Section 6.2.

This observation allows us to close the gap between RLVC and function approximation. It implies that the reinforcement learning step of RLVC (cf. line 6 of Algorithm 4.1) is not required to be implemented through RL algorithms for finite state-action spaces (cf. Section 4.4.1). Indeed, any RL algorithm that would use the linear approximation architecture above could be used to this end.

Generalization to Joint Classifiers

Starting from this observation about RLVC, it seems therefore natural to use function approximation techniques for reinforcement learning with a non-uniform discretization of the joint space. In RLJC, we propose to combine the Nonparametric Approximate Policy Iteration algorithm that was introduced in Section 6.3.1, with the state-action value functions constrained by $\mathcal{J}_k$ as an embedded approximation architecture.

The use of Nonparametric API requires the presence of a learning oracle $\mathcal{L}$ that is able to generate a constrained state-action value function from a database of samples $\langle s_t, a_t, v_t \rangle \in S \times A \times \mathbb{R}$ (cf. Definition 6.3). One plausible learning oracle simply consists in computing a constrained interpretation $\hat{Q}$ that averages the values of the samples over the joint classes that are defined by $\mathcal{J}_k$:

$$\hat{Q}(c) = \text{mean}\left\{ v_t \mid \mathcal{J}_k(s_t, a_t) = c \right\}, \tag{7.25}$$

for all the joint classes in C_k. The corresponding constrained state-action value function immediately follows from Equation 7.13. This process is summarized in Algorithm 7.5.

Accordingly, the computation of the optimal state-action value function Q^*_k that is constrained by $\mathcal{J}_k$ is achieved at line 6 of Algorithm 7.1 through a call to Algorithm 6.1. This successively leads to an optimal constrained interpretation $\hat{Q}^*_k$, that induces the optimal state-action value function Q^*_k. Note that Fitted Q Iteration [EGW05] could as well be used for this purpose.

7.4.4 Detecting and Removing Aliasing in the Joint Space

The algorithms that were presented for detecting aliasing and selecting new distinctive features are now adapted to the discretization of the joint class (cf. Sections 4.4.2 and 4.4.3). Thanks to Equation 7.17, the definition of Bellman residuals of Equation 4.22 can be further expanded:

$$\begin{aligned}
\Delta_t &= r_{t+1} + \gamma \sup_{a' \in A} Q^*_k(s_{t+1}, a') - Q^*_k(s_t, a_t) && (7.26)\\
&= r_{t+1} + \gamma \sup_{a' \in A} \hat{Q}^*_k(\mathcal{J}_k(s_{t+1}, a')) - \hat{Q}^*_k(\mathcal{J}_k(s_t, a_t)) && (7.27)\\
&= r_{t+1} + \gamma \max_{c \in C(s_{t+1})} \hat{Q}^*_k(c) - \hat{Q}^*_k(\mathcal{J}_k(s_t, a_t)), && (7.28)
\end{aligned}$$

Algorithm 7.5 — Learning a state-action value function that is constrained by $\mathcal{J}_k$

```
$\mathcal{L}(\{\langle s_t, a_t, v_t \rangle\})$ :-
    for $c \in C_k$ do
        $q[c] \leftarrow 0$
        $n[c] \leftarrow 0$
    end for
    for $t \leftarrow 1$ to $T$ do
        $c \leftarrow \mathcal{J}_k(s_t, a_t)$
        $q[c] \leftarrow q[c] + v_t$
        $n[c] \leftarrow n[c] + 1$
    end for
    for $c \in C_k$ do
        if $n[c] \neq 0$ then
            $q[c] \leftarrow q[c]/n[c]$
        end if
    end for
    return $Q : S \times A \mapsto \mathbb{R} : (s, a) \mapsto q[\mathcal{J}_k(s, a)]$
```

Algorithm 7.6 — Aliasing Criterion of RLJC

```
aliased$(c)$ :-
    $\Delta \leftarrow \{\Delta_t \mid \mathcal{J}_k(s_t, a_t) = c\}$
    if $\sigma^2(\Delta) > \tau$ then
        return true
    else
        return false
    end if
```

where $C(s_{t+1})$ is the set of compatible joint classes with respect to the percept s_{t+1}. As a consequence, the Bellman residuals can still be easily computed. Furthermore, because there is only a finite number of joint classes, it is possible to consider the set of the time stamps that are related to a given joint class $c \in C_k$:

$$T(c) = \{t \mid \mathcal{J}_k(s_t, a_t) = c\} \,. \tag{7.29}$$

If the database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ is not ill-conditioned, this set of time stamps is not reduced to a set containing at most one element, which would be the case if Equation 4.24 was directly used for continuous action spaces. Thus, the aliasing criterion for a given joint class $c \in C_k$ consists in testing whether the test of Bellman residuals

$$\{\Delta_t \mid t \in T(c)\} \tag{7.30}$$

has a variance that exceeds a threshold $\tau \in \mathbb{R}_0^+$. The resulting adaptation of Algorithm 4.2 is reported in Algorithm 7.6. The major difference with the aliasing criterion of RLVC is that there is no need for a loop over the actions.

Algorithm 7.7 — Feature selection process of RLJC

```
selector($c$) :–
    $F \leftarrow \mathcal{G}\left(\{(s_t, a_t) \mid t \in T(c)\}\right)$
    $f^* \leftarrow \perp$          {Best feature found so far}
    $r^* \leftarrow +\infty$        {Variance reduction induced by $f^*$}
    $T \leftarrow \{t \mid \mathcal{J}_k(s_t, a_t) = c\}$
    for joint features $f \in F$ do
        $S_\oplus \leftarrow \{\Delta_t \mid t \in T$ and $\mathcal{D}((s_t, a_t), f) = \textbf{true}\}$
        $S_\ominus \leftarrow \{\Delta_t \mid t \in T$ and $\mathcal{D}((s_t, a_t), f) = \textbf{false}\}$
        $p_\oplus \leftarrow |S_\oplus| / |T|$
        $p_\ominus \leftarrow |S_\ominus| / |T|$
        $r \leftarrow p_\oplus \cdot \sigma^2\left(S_\oplus\right) + p_\ominus \cdot \sigma^2\left(S_\ominus\right)$
        if $r < r^*$ and the distributions $(S_\oplus, S_\ominus)$ are significantly different then
            $f^* \leftarrow f$
            $r^* \leftarrow s$
        end if
    end for
    return $f^*$
```

Similarly, it is possible to define a set $F_k(c)$ of candidate features for a joint class $c \in C_k$ that is considered as aliased:

$$F_k(c) = \mathcal{G}\left(\{(s_t, a_t) \mid t \in T(c)\}\right), \tag{7.31}$$

where $\mathcal{G}$ is the joint feature generator. Note that this set may contain both perceptual and action features by virtue of Definition 7.5. Just like RLVC, RLJC selects the candidate joint feature inside $F_k(c)$ that most reduces the variance in the two sub-distributions of Bellman residuals that are induced by the joint feature. A Student's t-test is also applied, to make RLJC robust to non-deterministic environments. This leads to Algorithm 7.7, an adaptation of Algorithm 4.3 to non-uniform discretization of the action space. Note once again the removal of the loop over the actions, which is possible thanks to the definition of joint classes. This concludes the general description of RLJC.

7.5 Experimental Results

RLJC has been evaluated on a benchmark visual task that is very similar to that of Section 4.6.3. This task is depicted in Figure 7.4. Again, the goal of the agent is to reach as fast as possible one of the two exits of the maze. The only major difference is that the set of possible actions is continuous and not discrete anymore. At each location, the agent can make one step forward in any direction: The set A of actions is the continuous interval $[0°, 360°[$. Every move is altered by a Gaussian noise, whose standard deviation is 1% the size of the maze. Whenever a move would take the agent into a wall or outside the maze, its location is not changed.

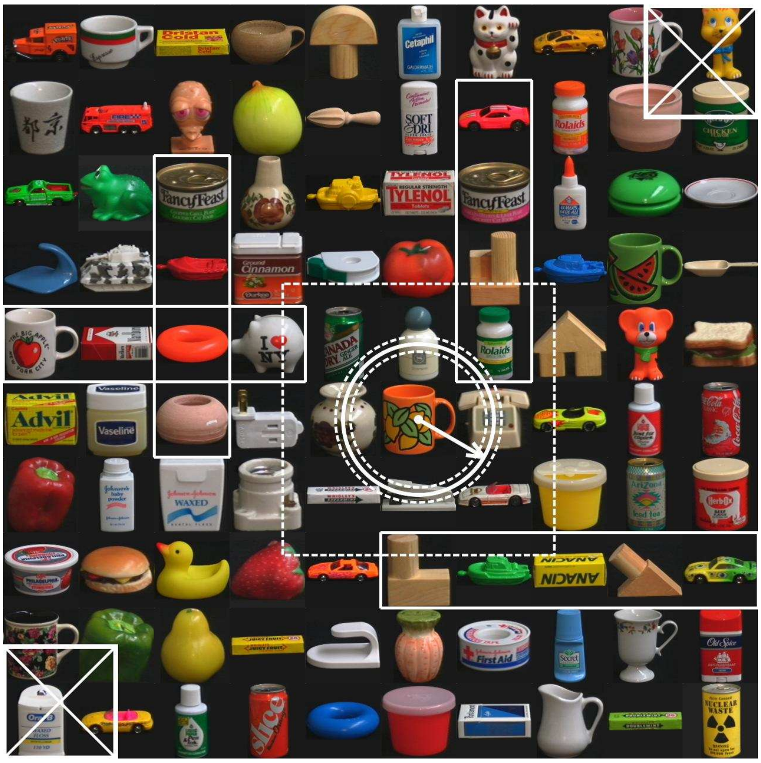


Figure 7.4: A continuous, noisy navigation task with continuous actions. The exits of the maze are indicated by boxes with a cross. Walls of glass are identified by solid lines. The agent is depicted at the center of the figure. The continuum of possible actions is represented by a solid circle. The two dashed circles indicate the standard deviation due to the noise. The sensors return a picture that corresponds to the dashed rectangular portion of the image.

In this experiment, SIFT visual features were used [Low04]. The entire tapestry includes 5520 interest points, leading to a subset of 2467 distinct visual features. The computation stopped when k reached 183, which took about 6 hours on a 3.0GHz Pentium IV using a database of 10,000 interactions that were collected by a fully randomized exploration policy. The final joint classifier $\mathcal{J}_k$ induces 896 joint classes, and tests the presence of 586 visual features and 309 action features. The greedy policy that results from this classifier is shown in Figure 7.5. Figure 7.6 compares the optimal value function of a discretized version of the problem with the one obtained through RLJC. The similarity between the two pictures indicates the soundness of our approach.

Interestingly, when applied to a similar task with only four *discrete* actions, RLVC generates a percept classifier $\mathcal{C}_k$ that contains 205 perceptual classes (cf. Section 4.6.3). In that case, $\mathcal{C}_k$ induces an optimal state-action value function that is parametrized by a matrix W of dimension $205 \times 4 = 820$ (cf. Section 7.4.3). This latter number is very close to the number of joint classes that is produced by RLJC (896). Therefore, discretizing the joint space produces a number of joint classes that corresponds to the underlying physical structure of the task.

7.6 Summary

This chapter has introduced *Reinforcement Learning of Joint Classes* (RLJC). RLJC is an extension of RLVC to high-dimensional, complex action spaces. RLJC adaptively discretizes the joint space of states and actions into a finite set of joint classes, by testing the presence of highly distinctive features. The homogeneous treatment of states and actions is at the same time elegant and powerful, and is conceptually similar to that of JoSTLe [MWSP04]. However, RLJC is more general than JoSTLe, in the sense that it can be applied to any perceptual space and to any action space upon which it is possible to define features. This notably includes visual input spaces, and continuous input/output spaces. Therefore, RLJC could learn mappings from *continuous* perceptual spaces to continuous action spaces as well. Experimental results on a simulated navigation task has indicated that RLJC is a promising framework for the interactive learning of visual tasks.

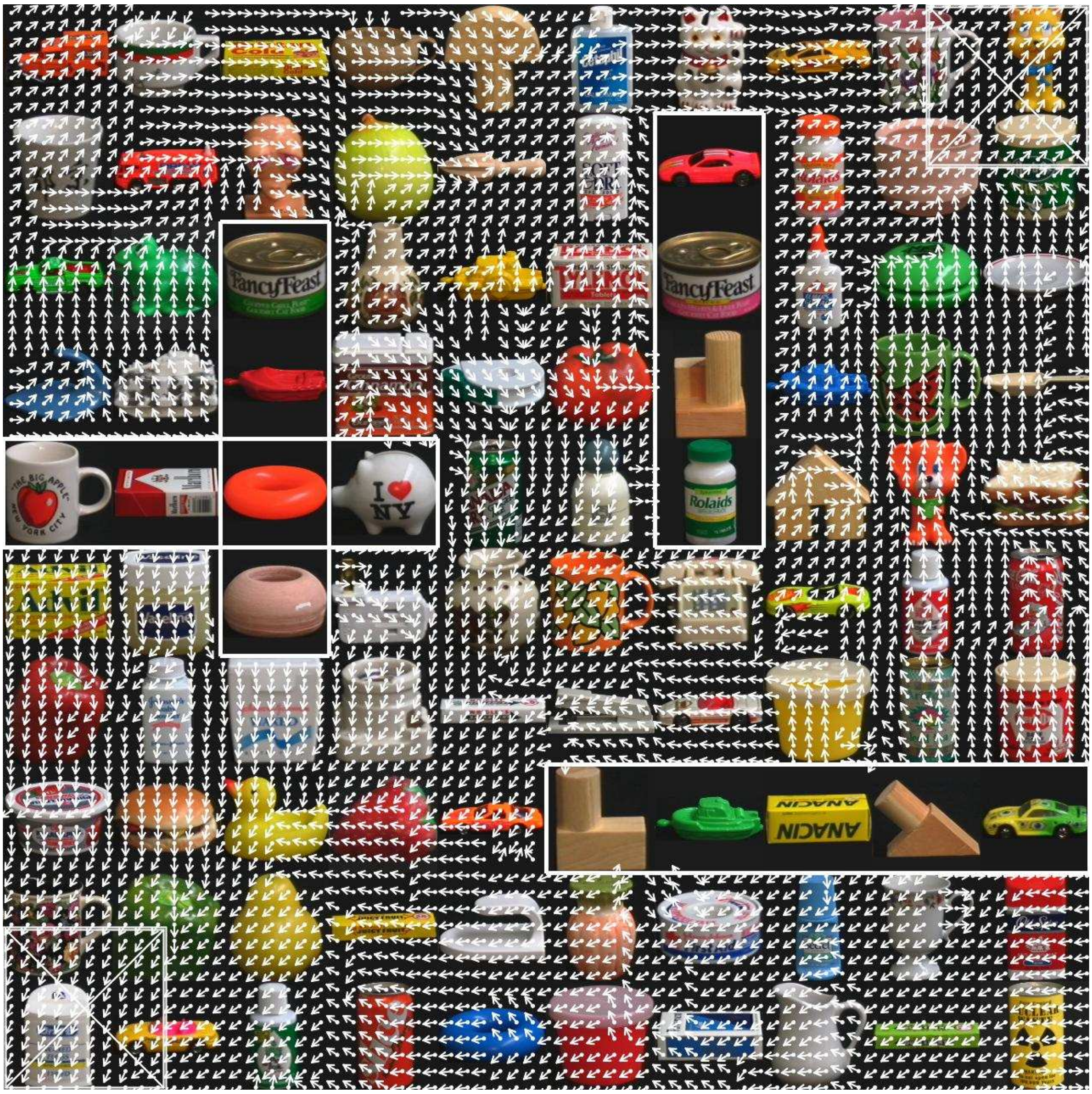


Figure 7.5: The resulting image-to-action mapping $\pi^* = \mathrm{argsup}_{a \in A}\, Q_k^*(s,a)$, sampled at regularly-spaced points. RLJC manages to choose the correct action at most locations.

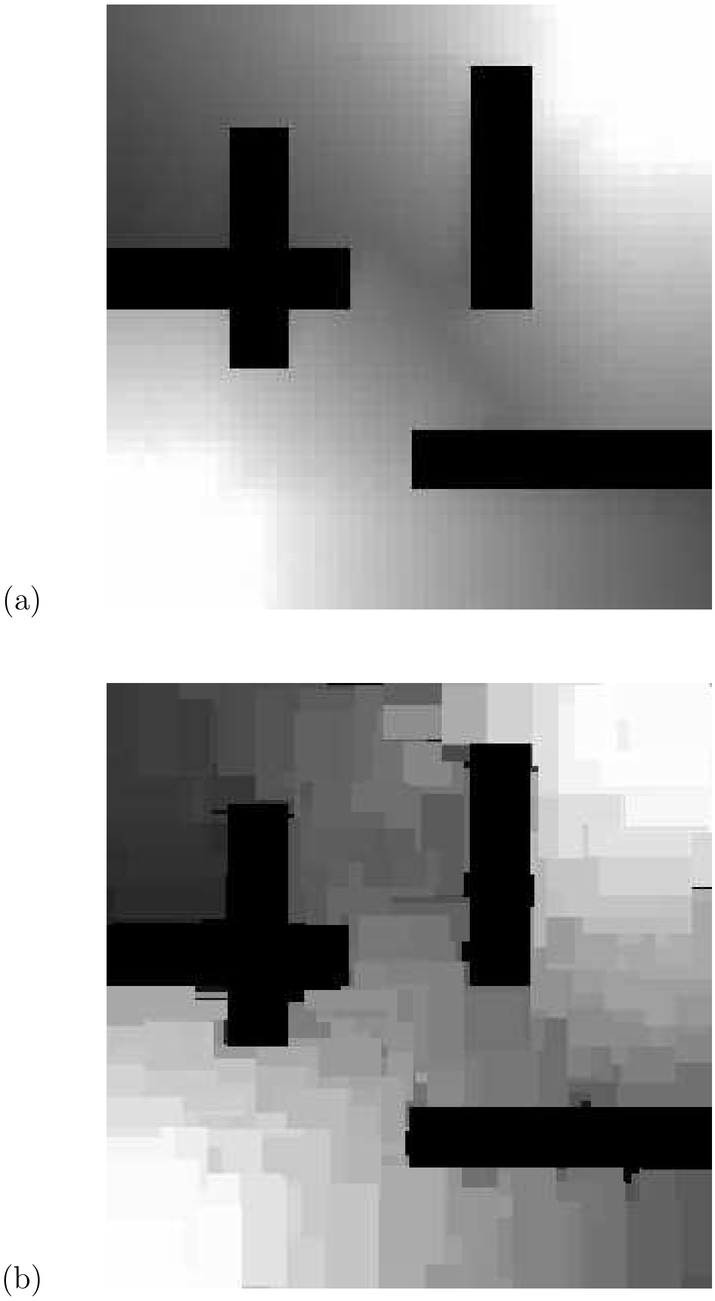


Figure 7.6: (a) The optimal value function, if the agent has direct access to its (x, y) position, if the set of possible locations is discretized into a 50 × 50 grid, and if the set of actions is discrete and contains 4 actions (go up, down, left or right). The brighter the location, the greater its utility. (b) The final value function obtained by RLJC.

CHAPTER

EIGHT

Conclusions and Perspectives

The opening chapters of this dissertation were devoted to different aspects of autonomous agents with visual sensors, and provided the context for the remaining chapters: Despite active and mature research in computer vision and artificial intelligence, designing robotic systems that can *autonomously learn* to solve visuomotor tasks is still an open, challenging and crucial problem. Accordingly, the central theme of this dissertation was defined as the design of algorithms for closed-loop learning of visual control policies.

The exploitation of reinforcement learning as an algorithmic framework to this end was motivated by the paradigm of purposive vision. The latter paradigm basically emphasizes the fact that a robotic agent should acquire visual skills from its *interactions* with an uncommitted environment in order to achieve some set of goals. In fact, there has been only relatively little focus on the learning aspects of such task-driven vision [Pia01]. The use of reinforcement learning naturally enables the fulfillment of a set of basic objectives that follow the philosophy of purposive vision (cf. Section 1.2). Notably, reinforcement learning algorithms solve a vision-for-action task by *interacting* with their environment. Moreover, they take into account the *dynamic, temporal dimension* of the surrounding environment, enabling them to solve reactive visual control tasks. The exact class of vision-for-action problems that were considered in this dissertation are those that can be modeled as Markov Decision Problems.

In this chapter, I will discuss and reassemble the conclusions from the previous chapters and make some suggestions for further research based on these conclusions.

8.1 Summary of the Contributions

The research work that is compiled in this dissertation has struggled with the intractability of standard reinforcement learning algorithms when applied to visuomotor tasks, because of the extremely high dimensionality of the visual percepts.

The proposed approach to this end relies on a novel idea, namely the exploitation of *local-appearance visual features* that are borrowed from the community of computer vision. Local-appearance vision summarizes images as finite sets of *visual features* by (1) seeking robust and highly informative patterns in the images through *interest point detectors*, then by (2) describing these patterns as vectors of real numbers through *local description techniques*. Recent results prove that this approach to computer vision is highly effective and fruitful.

Along this dissertation, two general, complementary ways for using visual features in the reinforcement learning process have been considered:

1. If an equivalence relation is defined over the space of visual features, a *percept classifier* can be used to partition the large, complex visual space into a few number of *perceptual classes*. If the latter set is kept small enough, any standard reinforcement learning can be used to extract an optimal image-to-action mapping. Therefore, the first main contribution of this dissertation has been to introduce general *adaptive-resolution methods* that can deal with any perceptual space upon which it is possible to define *perceptual features*, of which visual features are a particular instance.

   The resulting algorithm, *Reinforcement Learning of Visual Classes* (RLVC), incrementally refines a percept classifier by selecting perceptual features that are behaviorally important for solving the task, in a sequence of attempts to remove perceptual aliasing (cf. Chapter 4) [JP04, JP05b, JP05e, JP05c, JP06a]. RLVC can be thought of as a generalized version of McCallum's *utile distinctions* approach [McC96] that can sample features from a possibly infinite set thanks to the notion of *feature generators*. I have experimentally demonstrated that RLVC is successful at solving simulated visual navigation tasks.

2. The other investigated way to exploit visual features consists in using the raw content of the visual features, without ever considering an equivalence relation. In this framework, *feature regression models*, that associate the so-called *raw features* with a real-valued utility, are used as function approximators. I have proposed to embed such feature regression models inside a general, abstract version of the *Approximate Policy Iteration* architecture that was called *Nonparametric Approximate Policy Iteration*. The resulting algorithm, called *Visual Approximate Policy Iteration* (cf. Chapter 6) [JBP06], proved to be successful on a complex visual navigation task, at the cost of a higher computational expense than RLVC. Thus, Nonparametric and Visual Approximate Policy Iteration are two other important contributions of this research.

Another major contribution of this dissertation was the definition of *Reinforcement Learning of Joint Classes* (RLJC). RLJC is an extension of RLVC to complex, high-dimensional and/or continuous action spaces (cf. Chapter 7) [JP06b]. This algorithm is conceptually similar to RLVC, but it introduces *action features* that can be tested in the classifier, hereby producing a *non-uniform discretization* of the joint state-action space. RLJC features a homogeneous, elegant treatment of perceptual and action features. This algorithm has been used to solve a visual navigation task with continuous outputs. RLJC is a brand new, *general* approach to adaptive-resolution methods in reinforcement learning, for none of the previous algorithms in the literature can deal with *arbitrary, hybrid state-action spaces*. Thus, I argue that RLJC is potentially of major interest in the field of reinforcement learning.

Along the way, these main contributions have motivated the development of several additional tools that may prove to be useful, depending on the context of application:

1. Because of its greedy nature, RLVC is subject to overfitting. I have proposed a *forgetting mechanism* that provides RLVC with the ability to aggregate perceptual classes that share similar properties. This potentially enables RLVC to discard perceptual distinctions that subsequently prove to be useless (cf. Section 5.1) [JP05d, JP06a]. Technically, this modified version of RLVC uses sets of *Binary Decision Diagrams* (BDDs) [Bry86] as percept classifiers, instead of decision trees. I have shown that this technique indeed reduces overfitting effects.

2. It is possible to define vision-for-action tasks that cannot be solved using individual point features alone. This is for example the case when the agent has to take the distance between two visual patterns into account, or when the discriminative strength of the visual features is limited. In collaboration with F. Scalzo, I have proposed to iteratively construct a hierarchy of *spatial combinations of visual features* (cf. Section 5.2) [JSP05, JP06a]. A *composite feature* in this hierarchy is only triggered when two lower-level visual features are present at a given distance in the image, thus it is more discriminative than its subcomponents. This extension allows RLVC to deal with vision-for-action tasks that could not be solved otherwise, such as the visual car-on-the-hill control problem. From the point of view of computer vision, this is an interesting contribution, because it seems to be the first attempt to build hierarchies of visual features in a fully closed-loop and task-driven learning protocol.

3. In collaboration with C. Briquet, I have also shown how the construction of Extra-Trees can be distributed among a cluster of computers. Section 6.6 discussed how this idea was successfully implemented [JBP06]. This competitive advantage of Extra-Trees over other families of machine learning algorithms proved to be especially useful in the context of Visual Approximate Policy Iteration, because the latter algorithm constructs many Extra-Tree models from large databases. We have also proposed the protocol BitTorrent as an effective way to distribute a large amount of identical data from one master host to a set of slave hosts in a grid of computers. These ideas are certainly of primarily interest in the framework of distributed implementations of data mining algorithms. In particular, they should be directly applicable to any forest-based supervised learning algorithm, such as *Bagging* [Bre96] or *Random Forests* [Bre01].

It is worth pointing out that all the algorithms above have been implemented. Thus, a major part of this dissertation consists in the development of software and in the evaluation of the performance of the introduced algorithms on the benchmark tasks that were discussed along the dissertation.

Finally, I would like to point out other personal contributions that are not directly related to the core subject of this dissertation. These contributions are related to the domain of computer-aided verification [BJW01, Jod02, BJH03, BJW05], for which we have proposed to use automata-based approaches for the verification of hybrid systems with infinite state spaces; and to the domain of video-surveillance [BJV06], for which we have proposed to analyze and classify silhouettes through morphological size distributions and Classification Extra-Trees. Our distributed implementation of Extra-Tree was also used in the latter work.

8.2 Future Work

This dissertation will evidently not be the last word in the research on learning image-to-action mappings in a closed-loop protocol. There are many interesting topics that are suitable subjects for further research. This section brings together some of these topics that are, in my opinion, especially worth pursuing.

8.2.1 Non-Visual Control Problems

Although the proposed algorithms have all been designed with the objective of solving vision-for-action tasks in mind, care was taken in this dissertation to present them in a general version. This means that Reinforcement Learning of Visual Classes, Reinforcement Learning of Joint Classes as well as Nonparametric Approximate Policy Iteration could be used to solve non-visual control problems as well:

- Perceptual features that could be used in RLVC and in RLJC when percepts are continuous or when they consist of a set of bits were discussed in Sections 4.1.2 and 4.1.3;

- Action features that may prove to be useful in RLJC when an action associates a discrete value to a set of effector channels were proposed in Section 7.3.3;

- The notion of state-action value function approximator (cf. Section 6.3.1) potentially enables Nonparametric API to deal with non-visual perceptual spaces (and, in particular, with continuous spaces).

Thus, an important research direction would be to evaluate the performance of these algorithms on other combinations of perceptual and action spaces. Notably, comparing the performance of RLJC with that of the JoSTLe algorithm on a fully continuous task is of particular interest. From this perspective, another interesting open question is to test how well RLJC scales with respect to the dimensionality of the output action vector.

8.2.2 Implementation in Real Learning Robots

Future research also includes the demonstration of the applicability of these algorithms on a real robotic platform, such as visual navigation in a physical maze, or grasping objects by combining visual and haptic feedback [CPG01]. Of course, beyond the careful design of the robot itself, applying these algorithms directly on a real-world environment would raise practical problems, notably:

- *Hidden state problem.* The present work considers robotic agents with complete perception. In practice, most control problems feature partial observability, which means that their sensory feedback is insufficient to distinguish between any pair of percepts. Solving this problem would require the adaptation of RLVC and RLJC to Partially Observable Markov Decision Processes. An interesting research direction would be to provide the agent with a short-term memory that would enable it to remove ambiguities on its percepts. Following ideas proposed in McCallum's U Tree algorithm [McC96], RLVC and RLJC could be adapted to manage such a short-term memory by automatically selecting a history length from its interactions with the environment.

- *Non-stationary environments.* The dynamics of real environments as well as their sensory feedback continuously evolve over time. This is particularly true in visual tasks: For example, think about the changes in the lighting conditions along the day. This problem would also be exacerbated in *multi-agent setups*, in which each learning agent is free to change its own behavior as it learns, which results in fluctuations in the observed dynamics of the system. The algorithms should therefore automatically adapt to these changes in their environment. The techniques that have been developed to reduce overfitting effects in RLVC might prove to be especially relevant in this context. Indeed, a forgetting mechanism is needed to discard perceptual distinctions that become useless as the control task evolves.

- *Learning in real time.* The versions of RLVC and RLJC that have been proposed make batch processing, in that they first collect a database of interactions, then they exploit this knowledge to extract distinctive features. When dealing with a real robotic task, it is desirable to have algorithms that can learn in an *on-line fashion*, i.e. in *real time*, as they interact with the environment. This problem could be circumvented in RLVC and RLJC by splitting, every time a new interaction is collected, only the class that is related to this interaction. In this context, efficient, specialized data structures must be developed to represent databases of interactions. These specialized data structures should support a progressive evaluation of the variance of the Bellman residuals that occur in the various classes. The results of the feature generators should also be backed up so as to speed up the computations.

Addressing the same problems in the Visual Approximate Policy Iteration algorithm is more difficult, and is another open question that is worth being explored.

8.2.3 Enhancing the Proposed Algorithms

Even when dealing with simulated vision-for-action problems, the proposed algorithms could be improved in many ways. A first research direction is inspired by the work about Variable Resolution Grids [MM02], and would consist in evaluating the *influence* of areas of the perceptual space in order to select the classes that should be refined further. Another interesting question is whether the advantages of RLVC and V-API could be combined in an algorithm that would feature the low bias and variance of Extra-Trees together with the high speed of RLVC, while not requiring the use of a cluster of computers.

V-API should also be extended to continuous action spaces. At first, it would be worth evaluating the performance of V-API when the state-of-the-art simplex algorithm for general function maximization [NM65] is directly used to extract greedy actions (cf. Section 7.2). A distributed implementation of the simplex algorithm for general function maximization that would optimize approximators encoded as an Extra-Tree model would be extremely desirable in this context. From an implementation perspective, also note that the current distributed implementation of the learning of Extra-Trees does not feature task preemption. Thus, future work could also focus on producing a gridified version of our software for learning Extra-Trees, that would be able to handle the load inbalance. This would require the use of more complex task scheduling policies.

Finally, as far as RLJC is concerned, it would be extremely promising to combine the joint classifiers with *interpolation methods* when dealing with joint state-action spaces that have either a continuous perceptual space, or a continuous action space. In such a setup, the utility of a state-action pair could vary through interpolation in the joint class that corresponds to this pair. This idea should combine the respective advantages of Variable Resolution Grids [MM02] and RLJC, and should allow the definition of a competitive reinforcement learning algorithm for hybrid state-action spaces.

8.2.4 Towards Better Visual Features

From the point of view of the visual features themselves, the use of image patches at random locations [Mar05, MGPW05] is very promising for solving vision-for-action tasks through V-API [EMW06], and should be investigated in future work.

A direction that I did not follow but that should be promising when dealing with real, robotic vision-for-action tasks would be to combine different kinds of local-appearance visual features into a heterogeneous visual feature space. The respective discriminative skills of these families of features would then combine and reinforce each other. For instance, combining visual features that are detected through a Harris-based detector with visual features obtained by a Hessian-based detector would capture both discontinuous and uniform patterns in the visual percepts, which might be useful to avoid hidden state problems. Integrating such multi-modal visual features in the RLVC and in the V-API frameworks should be quite natural. In a similar spirit, cues from other senses than vision (haptics, sounds, smells. . .) could be integrated in multimodal sensory signals [FAT05].

In the context of learning spatial combinations of visual features, it would be very interesting to take the *relative orientations* between pairs of lower-level visual features into account in composite features, instead of the distance alone. This idea is pursued by Scalzo and Piater in an unsupervised learning framework [SP06]. To this end, clustering in the joint space of distance and relative orientations would be required. Finally, a major imperfection of the current scheme for closed-loop generation of geometric combinations of features is that whenever a subcomponent is not detected (for example because of noise in the image), the detection of higher-level features that depend on it is never triggered. As a consequence, hierarchies of visual features are so far only useful in strongly controlled experimental setups. To state it more intuitively, the current learning architecture is "too deterministic." A *probabilistic* method for representing such hierarchies would also be highly desirable. As in the work by Scalzo and Piater [SP06], the detection of composite features should be inferred through evidence propagation [SIFW03], which would allow the detection of composite features even though all of their subcomponents are not readily detected in the images.

BIBLIOGRAPHY

[Alb75] J.S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97(3):220–227, September 1975. 124

[All84] P.K. Allen. Surface descriptions from vision and touch. In *Proc. of the Conference on Robotics*, pages 394–397, Atlanta (GA, USA), March 1984. IEEE. 9

[Alo90] Y. Aloimonos. Purposive and qualitative active vision. In *Proc. of the 10th International Conference on Pattern Recognition*, pages 436–460, June 1990. 4

[AMC00] B.S. Anderson, A.W. Moore, and D. Cohn. A nonparametric approach to noisy and costly optimization. In *Proc. of the 7th International Conference on Machine Learning*, pages 17–24, Stanford University (CA, USA), June 2000. 149

[AMS97] C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997. 125

[AN05] P. Abbeel and A. Ng. Exploration and apprenticeship learning in reinforcement learning. In L. De Raedt and S. Wrobel, editors, *Proc. of the 22nd International Conference on Machine Learning*, pages 1–8, Bonn (Germany), August 2005. ACM. 38

[And87] C.W. Anderson. Strategy learning in multilayer connectionist representations. In *Proc. of the 4th International Workshop on Machine Learning*, pages 103–114, Irvine (CA, USA), June 1987. 127

[ANTH94] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In *Proc. of IAPR/IEEE Workshop on Visual Behaviors*, pages 112–118, 1994. 8

[AVAS04] N. Abe, N. Verma, D. Apte, and R. Schroko. Cross channel optimized marketing by reinforcement learning. In *Proc. of the 10th International*

Conference on Knowledge Discovery and Data Mining, pages 767–772, New York, NY, USA, 2004. ACM Press. 38

[AWB88] Y. Aloimonos, I. Weiss, and A. Bandopadhay. Active vision. *International Journal of Computer Vision*, 1(4):333–356, 1988. 5

[Bai95] Leemon C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proc. of the 12th International Conference on Machine Learning*. Morgan Kaufmann, 1995. 126

[Baj88] R.K. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):996–1005, August 1988. 5

[Bar95] A.G. Barto. Adaptive critics and the basal ganglia. In J.C. Houk, J.L. Davis, and D.G. Beiser, editors, *Models of Information Processing in the Basal Ganglia*, pages 215–232. MIT press, Cambridge, MA, 1995. 36

[Bar06] O. Barnich. Détection de personnes dans une séquence vidéo, 2006. DEA Thesis, University of Liège. 138

[Bau00] A. Baumberg. Reliable feature matching across widely separated views. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 774–781, Hilton Head (SC, USA), June 2000. 57

[BB92] D.H. Ballard and C.M. Brown. Principles of animate vision. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 56(1):3–21, July 1992. 5

[BB96] S.J. Bradtke and A.G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1–3):33–57, 1996. 127, 130

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems (extended abstract). In *Proc. of the 9th International Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 167–178, Haifa (Israel), June 1997. Springer. 99

[BBS95] A.G. Barto, S.J. Bradtke, and S.P. Singhe. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995. 38

[BCGM98] S. Belongie, C. Carson, H. Greenspan, and J. Malik. Color and texture based image segmentation using the expectation-maximization algorithm and its application to content-based image retrieval. In *Proc. of the 6th International Conference on Computer Vision*, pages 675–682, Bombay (India), January 1998. Narosa Publishing House. 44

[BD94] A.G. Barto and M. Duff. Monte-Carlo matrix inversion and reinforcement learning. In J.D. Cohen, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, pages 687–694, San Francisco, 1994. Morgan Kaufmann. 37

[Bd06] C. Briquet and P.-A. de Marneffe. What is the grid? tentative definitions beyond resource coordination. Technical report, University of Liège, Liège (Belgium), 2006. 141, 142

[BDG00] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000. 73

[Bea78] P.R. Beaudet. Rotationally invariant image operators. In *Proc. of the 4th International Joint Conference on Pattern Recognition*, pages 579–583, Tokyo, 1978. 57

[Bel57a] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957. 16, 21, 26, 71

[Bel57b] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957. 16

[Ber95] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 1995. 21

[Ber00] D.P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 2 edition, 2000. 21

[BFG$^+$93a] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *International Conference on Computer-Aided Design*, pages 188–192, 1993. 73

[BFG$^+$93b] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proc. of International Conference on Computer-Aided Design*, pages 188–192, 1993. 99, 110

[BFS84] L. Breiman, J.H. Friedman, and C.J. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984. 69, 77, 83, 84, 98, 134

[BJH03] B. Boigelot, S. Jodogne, and F. Herbreteau. Hybrid acceleration using real vector automata. In *Proc. of the 15th International Conference on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 193–205, Boulder (CO, USA), July 2003. Springer-Verlag. 99, 170

[BJV06] O. Barnich, S. Jodogne, and M. Van Droogenbroeck. *Robust Analysis of Silhouettes by Morphological Size Distributions*, volume Advanced Concepts for Intelligent Vision Systems (ACIVS 2006) of *Lecture Notes on Computer Science*, page 12. Springer Verlag, Antwerpen, September 2006. Accepted for publication. 138, 170

[BJW01] B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Proc. International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625, Sienna (Italy), June 2001. Springer-Verlag. 99, 170

[BJW05] B. Boigelot, S. Jodogne, and P. Wolper. An effective decision procedure for linear arithmetic with integer and real variables. *ACM Transactions on Computational Logic (TOCL)*, 6(3):614–633, July 2005. 99, 170

[BK93] L.C. Baird and A.H. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical report, Wright-Patterson Air Force Base Ohio: Wright Laboratory, 1993. WL-TR-93-1147. 149

[Blu67] H. Blum. A transformation for extracting new descriptors of shape. In *Proc. of the Symposium on Models for the Perception of Speech and Visual Form*, pages 362–380, 1967. 43

[BM95] J. Boyan and A. Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems*, 7:369–376, 1995. 128

[BMP02] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 2002. 62

[Boi99] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, University of Liège, Liège (Belgium), 1999. 99

[Boy02] J.A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2–3):233–246, 2002. 127

[BP79] A. Berman and R.J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press, New York, 1979. 211

[Bre96] L. Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996. 134, 169

[Bre01] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 134, 141, 169

[Bri90] J.S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in Neural Information Processing Systems*, volume 2, pages 211–217. Morgan Kaufmann, 1990. 34

[BRW98] B. Boigelot, S. Rassart, and P. Wolper. On the expressiveness of real and integer arithmetic automata (extended abstract). In *Proc. of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 152–163, Aalborg (Denmark), July 1998. Springer. 99

[Bry86] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions in Computers*, 8(35):677–691, 1986. 99, 100, 102, 169

[Bry92] R.E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992. xvii, 73, 99, 100

[BS01] J. Bagnell and J. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proc. of the International Conference on Robotics and Automation*. IEEE, 5 2001. 38

[BSA83] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13(5):835–846, 1983. 29, 35

[BT89] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989. 23, 27, 28

[BT96] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996. xvii, 6, 15, 26, 33, 35, 122, 125, 126, 127, 129

[BT05] G. Bouchard and B. Triggs. Hierarchical part-based visual object categorization. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 710–715, San Diego (CA, USA), June 2005. 111, 120

[BT06] BitTorrent. Wikipedia, the Free Encyclopedia, 2006.
http://en.wikipedia.org/wiki/Bittorrent. 142

[BTT06] BitTorrent tracker. Wikipedia, the Free Encyclopedia, 2006.
http://en.wikipedia.org/wiki/Bittorrent_tracker. 142

[BTVG06a] H. Bay, T. Tuytelaars, and L. Van Gool. Implementation of SURF (Speeded Up Robust Features), 2006.
http://www.vision.ee.ethz.ch/~surf/. 63

[BTVG06b] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. In *Proc. of the 9th European Conference on Computer Vision*, pages 404–417, May 2006. 58, 62, 63

[BTW98] J. Baxter, A. Trigdell, and L. Weaver. Knightcap: A chess program that learns by combining TD(λ) with game-tree search. In *Proc. of the 15th International Conference on Machine Learning*, pages 28–36. Morgan Kaufmann, 1998. 38

[BVZ01] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001. 44

[BY92] A. Blake and A. Yuille, editors. *Active Vision*. MIT Press, Cambridge (MA, USA), 1992. 5

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986. 43, 62

[CB98] R.H. Crites and A.G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235–42, 1998. 38

[CBGM02] C. Carson, S. Belongie, S. Greenspan, and J. Malik. BlobWorld: Image segmentation using expectation-maximization and its application to image querying. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8):1026–1038, 2002. 44

[CBS$^+$03] W. Cirne, F. Brasileiro, J. Sauve, N. Andrade, D. Paranhos, E. Santos-Neto, and R. Medeiros. Grid computing for bag of tasks applications. In *Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and E-Goverment*, September 2003. 140

[CdVC98] V. Colin de Verdière and J.L. Crowley. Visual recognition using local appearance. In *Proc. of the 5th European Conference on Computer Vision*, volume 1 of *Lecture Notes in Computer Science*, pages 640–654. Springer, June 1998. 59

[CH67] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. 125

[Chr92] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *National Conference on Artificial Intelligence*, pages 183–188, 1992. 74, 75

[Cic95] P. Cichosz. Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *Journal on Artificial Intelligence*, 2:287–318, 1995. 37

[CJG97] J.A. Coelho, A. Jefferson, and R.A. Grupen. A control basis for learning multifingered grasps. *Journal of Robotic Systems*, 14(7):545–557, 1997. 9

[CK91] D. Chapman and L.P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proc. of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 726–731, Sydney, August 1991. 69, 74, 75, 84, 85

[CM95] J.L. Crowley and J. Martin. Experimental comparison of correlation techniques. In *Proc. of the International Conference on Intelligent Autonomous Systems*, March 1995. 44, 64

[Coh03] B. Cohen. Incentives build robustness in BITTORRENT. In *Proc. of the Workshop on Economics of Peer-to-Peer Systems*, 5 2003. 11, 142

[Cou02] R. Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002. 127

[Cou03] R. Coulom. Model-based actor-critic algorithm in continuous time and space. In *Proc. of the 6th European Workshop on Reinforcement Learning*, Nancy (France), 2003. 31

[CPG01] J.A. Coelho, J.H. Piater, and R.A. Grupen. Developing haptic and visual perceptual categories for reaching and grasping with a humanoid robot. *Robotics and Autonomous Systems, special issue on Humanoid Robots*, 37(2–3):195–218, 2001. 9, 171

[CRS94] P.S. Churchland, V.S. Ramachandran, and T.J. Sejnowski. A critique of pure vision. In C. Koch and J.L. Davis, editors, *Large Scale Neuronal Theories of the Brain*, pages 23–60. MIT Press, Cambridge (MA, USA), 1994. 2, 3

[CTB$^+$99] C. Carson, M. Thomas, S. Belongie, J.M. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. In *Proc. of the 3rd International Conference on Visual Information Systems*, volume 1614 of *Lecture Notes in Computer Science*, pages 509–516. Springer, 1999. 46

[CWN04] B. Caputo, C. Wallraven, and M.-E. Nilsback. Object categorization via local kernels. In *Proc. of the 17th International Conference on Pattern Recognition*, pages 132–135, August 2004. 65

[Dav98] A.J. Davison. *Mobile Robot Navigation using Active Vision*. PhD thesis, University of Oxford, 1998. 5

[Day92] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362, 1992. 37

[dCS$^+$04] F.A.B. da Silva, S. Carvalho, H. Senger, E.R. Hruschka, and C.R.G. de Farias. Running data mining applications on the grid: A bag-of-tasks approach. In *Proc. of the International Conference on Computational Science and its Applications*, volume 3044 of *Lecture Notes in Computer Science*, pages 168–177. Springer, May 2004. 140

[Der70] C. Derman. *Finite State Markovian Decision Processes.* Academic Press, New York, 1970. 21, 22, 24

[Deu04] H. Deubel. Localization of targets across saccades: Role of landmark objects. *Visual Cognition*, 11(2–3):173–202, February 2004. 5

[DEVW06] R. Dardenne, J.-J. Embrechts, M. Van Droogenbroeck, and N. Werner. A video-based human-computer interaction system for audio-visual immersion. In *Proc. of the 2nd Annual IEEE BENELUX/DSP Valley Signal Processing Symposium (SPS-DARTS)*, pages 23–26, Antwerp (Belgium), March 2006. IEEE Benelux.

[DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989. 73

[DLR77] A.P. Dempster, N.M. Laird, and S.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society (Series B Methodological)*, 39(1):1–38, 1977. 44

[Dor05] G. Dorko. Implementation of various detectors and descriptors, 2005. http://lear.inrialpes.fr/people/dorko/downloads.html. 63

[Doy96] K. Doya. Temporal difference learning in continuous time and space. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1073–1079. MIT Press, 1996. 31

[Doy00] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12:243–269, 2000. 31

[DR00] G. Delzanno and J.-F. Raskin. Symbolic representation of upward closed sets. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 426–440, Berlin (Germany), March 2000. 99

[Ebe96] D. Eberly. *Ridges in Image and Data Analysis.* Kluwer Academics Publishers, 1996. 43

[EC04] J. Eichhorn and O. Chapelle. Object categorization with SVM: Kernels for local features. Technical Report 137, Max Planck Institute for Biological Cybernetics, 07 2004. 59, 65

[EGW03] D. Ernst, P. Geurts, and L. Wehenkel. Iteratively extending time horizon reinforcement learning. In *Proc. of the 14th European Conference on Machine Learning*, pages 96–107, Dubrovnik (Croatia), September 2003. 116

[EGW05] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, April 2005. 9, 11, 82, 121, 128, 130, 134, 136, 137, 138, 145, 149, 159

[EM95] F. Ennesser and G.G. Medioni. Finding Waldo, or focus of attention using local color information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):805–809, 1995. 59

[EMW06] D. Ernst, R. Marée, and L. Wehenkel. Reinforcement learning with raw pixels as state input. In *Proc. of the International Workshop on Intelligent Computing in Pattern Analysis/Synthesis*, Xi'An, China, August 2006. Accepted for publication. 9, 140, 172

[Ern03] D. Ernst. *Near Optimal Closed-Loop Control Application to Electric Power Systems*. PhD thesis, University of Liège, Liège (Belgium), 2003. 38

[ESGW06] D. Ernst, G.B. Stan, J. Goncalves, and L. Wehenkel. Clinical data based optimal STI strategies for HIV: A reinforcement learning approach. In *Proc. of Benelearn 2006*, 5 2006. 38

[EU05] B. Epshtein and S. Ullman. Feature hierarchies for object classification. In *Proc. of the 10th IEEE International Conference on Computer Vision*, pages 220–227, Beijing (China), October 2005. 111

[FA91] W.T. Freeman and E.H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, September 1991. 60

[FA95] C. Fermüller and Y. Aloimonos. Vision and action. *Image and Vision Computing*, 13(10):725–745, December 1995. 5

[FAT05] P. Fitzpatrick, A. Arsenio, and E.R. Torres-Jara. Reinforcing robot perception of multi-modal events through repetition and redundancy and repetition and redundancy. *Interaction Studies Journal*, 2005. Accepted for publication. 173

[FB81] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 43

[FH05] P.F. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, January 2005. 111

[FHI99] D.A. Forsyth, J.A. Haddon, and S. Ioffe. Finding objects by grouping primitives. In *Shape, Contour and Grouping in Computer Vision*, pages 302–318, London (UK), 1999. Springer-Verlag. 111

[FK04] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004. 141, 142

[FP03] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003. 43, 56, 61, 111

[FP05] L. FeiFei and P. Perona. A Bayesian hierarchical model for learning natural scene categories. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 524–531, 2005. 52

[FPZ03] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, Madison (WI, USA), June 2003. 120

[FS04] P. Fidelman and P. Stone. Learning ball acquisition on a physical robot. In *Proc. of the International Symposium on Robotics and Automation*, August 2004. 38

[FtHRKV91] L. Florack, B. ter Haar Romeny, J. Koenderink, and M. Viergever. General intensity transformations and second order invariants. In *Proc. of the 7th Scandinavian Conference on Image Analysis*, pages 338–345, 1991. 61

[Gas02] C. Gaskett. *Q-Learning for Robot Control*. PhD thesis, Research School of Information Sciences and Engineering, Australian National University, 2002. 149

[GB01] V. Gouet and N. Boujemaa. Object-based queries using color points of interest. In *IEEE Workshop on Content-Based Access of Image and Video Libraries*, pages 30–36, Kauai (HI, USA), 2001. 54, 93, 117

[GB02] V. Gouet and N. Boujemaa. About optimal use of color points of interest for content-based image retrieval. Technical Report RR-4439, INRIA Rocquencourt, Le Chesnay (France), April 2002. 54, 61

[GDG03] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1–2):163–223, 2003. 110

[GEW06] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 36(1):3–42, 2006. 9, 11, 12, 13, 54, 65, 121, 134, 135, 136, 137, 141, 143, 145

[GFZ00] C. Gaskett, L. Fletcher, and A. Zelinsky. Reinforcement learning for visual servoing of a mobile robot. In *Proc. of the Australian Conference on Robotics and Automation*, Melbourne (Australia), August 2000. 8

[GHPV05] P. Gabriel, J-B. Hayet, J.H. Piater, and J. Verly. Object tracking using color interest points. In *Proc. of the International Conference on Advanced Video and Signal Based Surveillance*, pages 159–164, 2005. 54, 61, 111

[Gil98] S. Gilles. *Robust Description and Matching of Images*. PhD thesis, University of Oxford, 1998. 58

[GL86] W.E.L. Grimson and T. Lozano-Pérez. Model-based recognition and localization from tactile data. *Journal of Robotics Research*, 3(3):3–35, 1986. 9

[GL96] J. Gårding and T. Lindeberg. Direct computation of shape cues based on scale-adapted spatial derivative operators. *International Journal of Computer Vision*, 17(2):163–191, 1996. 57

[Gor95] G.J. Gordon. Stable function approximation in dynamic programming. In *Proc. of the International Conference on Machine Learning*, pages 261–268, 1995. 116, 127

[Gor99] G.J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie Mellon University, Pittsburgh, June 1999. 127

[Gor01] G.J. Gordon. Reinforcement learning with function approximation converges to a region. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems*, pages 1040–1046. MIT Press, 2001. 126

[GS83] E.J. Gibson and E.S. Spelke. The development of perception. In John H. Flavell and Ellen M. Markman, editors, *Handbook of Child Psychology Vol. III: Cognitive Development*, chapter 1, pages 2–76. Wiley, 4th edition, 1983. 3

[GSK98] H.-M. Gross, V. Stephan, and M. Krabbes. A neural field approach to topological reinforcement learning in continuous action spaces. In *Proc. of the IEEE World Congress on Computational Intelligence*, volume 3, pages 1992–1997, 1998. 148

[GSM03] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: A texture classification example. In *Proc. of the 9th IEEE International Conference on Computer Vision*, pages 456–463, Nice (France), October 2003. IEEE Computer Society. 65

[Gue03] C. Guestrin. *Planning under Uncertainty in Complex Structured Environments*. PhD thesis, Stanford University, August 2003. 73

[GWZ99] C. Gaskett, D. Wettergreen, and A. Zelinsky. Q-learning in continuous state and action spaces. In *Australian Joint Conference on Artificial Intelligence*, pages 417–428, 1999. 149

[HAB95] J.C. Houk, J.L. Adams, and A.G. Barto. A model of how the basal ganglia generate and use neural signals that predict reinforcement. In J.C. Houk, J.L. Davis, and D.G. Beiser, editors, *Models of Information Processing in the Basal Ganglia*, pages 249–270. MIT press, Cambridge, MA, 1995. 36

[Has03] S.W. Hasinoff. Reinforcement learning for problems with hidden state, 2003. Technical Report, University of Toronto, Department of Computer Science. 74

[HG98] M. Huber and R. Grupen. A control structure for learning locomotion gaits. In *7th Int. Symposium on Robotics and Applications*, Anchorage (AK, USA), May 1998. TSI Press. 38

[Hin70] K. Hinderer. *Foundation of Non-Stationary Dynamic Programming with Discrete Time Parameter*, volume 33 of *Lecture Notes in Operations Research and Mathematical Systems*. Springer-Verlag, Berlin, 1970. 21

[HLL96] O. Hernández-Lerma and B. Lasserre. *Discrete-Time Markov Control Processes*. Springer, New York, 1996. 21

[HLL99] O. Hernández-Lerma and B. Lasserre. *Further Topics on Discrete-Time Markov Control Processes*. Springer, New York, 1999. 21

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975. 149

[How60] R.A. Howard. *Dynamic Programming and Markov Processes*. Technology Press and Wiley, Cambridge (MA) and New York, 1960. 21, 26, 27, 28

[How71] R.A. Howard. *Dynamic Probabilistic Systems*, volume 2. Wiley, 1971. 21

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. of the 4th Alvey Vision Conference*, pages 147–151, University of Manchester (UK), August 1988. 47, 48, 50

[HSHB99] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proc. of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 279–288, Stockholm (Sweden), 1999. 73

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, may 1998. 65

[ISS02] M. Iida, M. Sugisaka, and K. Shibata. Direct-vision-based reinforcement learning to a real mobile robot. In *Proc. of International Conference of Neural Information Processing Systems*, volume 5, pages 2556–2560, 2002. 8, 11

[JBP06] S. Jodogne, C. Briquet, and J.H. Piater. Approximate policy iteration for closed-loop learning of visual tasks. In *Proc. of the 17th European Conference on Machine Learning (ECML)*, volume 4212 of *Lecture Notes in Computer Science*, pages 222–233, September 2006. xvii, 9, 11, 140, 146, 168, 169

[JJS94] T. Jaakkola, M.I. Jordan, and S.P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 703–710. Morgan Kaufmann Publishers, 1994. 33

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999. 114

[Jod01] S. Jodogne. Représentation à états finis d'ensembles de réels, May 2001. Master's thesis, University of Liège. 99

[Jod02] S. Jodogne. Automata-based representations for the verification of hybrid systems. In *Proc. Summer School Modelling and Verification of Parallel Processes (MOVEP)*, pages 320–325, Nantes (France), June 2002. 99, 170

[Jod05] S. Jodogne. Montefiore image database, 2005. The content of this database can be previewed at the following address:
`http://www.montefiore.ulg.ac.be/~jodogne/phd-database/`. 53, 105, 108

[JP04] S. Jodogne and J.H. Piater. Interactive selection of visual features through reinforcement learning. In M. Bramer, F. Coenen, and

T. Allen, editors, *Proc. of the 24th SGAI Internal Conference on Innovative Techniques and Applications of Artificial Intelligence*, volume 21 of *Research and Development in Intelligent Systems*, pages 285–298, Cambridge (UK), December 2004. Springer-Verlag. 11, 71, 168

[JP05a] S. Jodogne and J.H. Piater. Apprentissage interactif de liaisons directes entre perceptions visuelles et actions. In *Actes du Congrès ORASIS*, Fournols (France), May 2005. 11

[JP05b] S. Jodogne and J.H. Piater. Controlling an agent by focusing its attention on interactively selected patterns. *Belgian Journal of Electronics and Communications, special issue: URSI Forum 2004*, 1:14–16, 2005. 11, 168

[JP05c] S. Jodogne and J.H. Piater. Interactive learning of mappings from visual percepts to actions. In L. De Raedt and S. Wrobel, editors, *Proc. of the 22nd International Conference on Machine Learning (ICML)*, pages 393–400, Bonn (Germany), August 2005. ACM. xvii, 11, 71, 84, 168

[JP05d] S. Jodogne and J.H. Piater. Learning, then compacting visual policies (extended abstract). In *Proc. of the 7th European Workshop on Reinforcement Learning (EWRL)*, pages 8–10, Napoli (Italy), October 2005. 11, 97, 169

[JP05e] S. Jodogne and J.H. Piater. Reinforcement learning of perceptual classes using Q-learning updates. In M.H. Hamza, editor, *Proc. of the 23rd IASTED International Multi-Conference on Artificial Intelligence and Applications*, pages 445–450, Innsbruck (Austria), February 2005. Acta Press. 11, 71, 168

[JP06a] S. Jodogne and J.H. Piater. Reinforcement learning of visual control policies. *Journal of Artificial Intelligence Research*, 2006. Accepted for publication. 11, 168, 169

[JP06b] S. Jodogne and J.H. Piater. Task-driven discretization of the joint space of visual percepts and continuous actions. In *Proc. of the 17th European Conference on Machine Learning (ECML)*, volume 4212 of *Lecture Notes in Computer Science*, pages 210–221, September 2006. Accepted for publication. xvii, 11, 168

[JS04] F. Jurie and C. Schmid. Scale-invariant shape features for recognition of object categories. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 90–96, Washington (DC, USA), June 2004. 47

[JSP05] S. Jodogne, F. Scalzo, and J.H. Piater. Task-driven learning of spatial combinations of visual features. In *Proc. of the IEEE Workshop* *on Learning in Computer Vision and Pattern Recognition*, San Diego (CA, USA), June 2005. IEEE. 11, 97, 169

[JV96] A.K. Jain and A. Vailaya. Image retrieval using color and shape. *Pattern Recognition*, 29(8):1233–1244, 1996. 46

[KA97] R. Kretchmar and C. Anderson. Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In *Proc. of the IEEE International Conference on Neural Networks*, pages 834–837, Houston (TX, USA), 1997. 126

[Kae90] L. P. Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University, 1990. 35

[Kal60] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering, Transactions of the ASME*, 82(1):35–45, 1960. 44

[KBTD95] J. Konczak, M. Borutta, H. Topka, and J. Dichgans. Development of goal-directed reaching in infants: Hand trajectory formation and joint force control. *Experimental Brain Research*, 106:156–168, 1995. 3

[KF04] C. Kwok and D. Fox. Reinforcement learning for sensing strategies. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, 2004. 8

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KHW95] N. Kushmerick, S. Hanks, and D.S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995. 73

[Kir01] D. Kirkove. S'entendre à travers nos cultures, June 2001. Master's thesis, Institut Sainte-Julienne (Liège).

[KJ92] R. Kergen and P. Jodogne. Computerized control of the blankholder pressure on deep drawing presses. Technical Report 920433, Society of Automotive Engineers, Warrendale (PA, USA), 1992.

[KLC98] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. 74

[KLG$^+$05] M. Khamassi, L. Lachèze, B. Girard, A. Berthoz, and A. Guillot. Actor-critic models of reinforcement learning in the basal ganglia: From natural to artificial rats. *Adaptive Behavior, Special Issue Towards Artificial Rodents*, 13(2):131–148, 2005. 36, 39

[KLM96] L.P. Kaelbling, M.L. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. xvii, 6, 15, 24, 33, 35, 38, 148

[KMND04] D. Keysers, W. Macherey, H. Ney, and J. Dahmen. Adaptation in statistical pattern recognition using tangent vectors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):269–274, 2004. 64

[KS60] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Van Nostrand, Princeton, 1960. 212

[KS98] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proc. of the 15th International Conference on Machine Learning*, pages 260–268. Morgan Kaufmann, 1998. 38

[KS04a] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 506–513, Washington (DC, USA), June 2004. 62, 64

[KS04b] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 2619–2624, New Orleans, 5 2004. 38

[KT03] V.R. Konda and J.N. Tsitsiklis. Actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2003. 29

[KTZ04] M.P. Kumar, P.H.S. Torr, and A. Zisserman. Extending pictorial structures for object recognition. In *Proc. of the British Machine Vision Conference*, 2004. 111

[KV86] P.R. Kumar and P. Varaiya. *Stochastic Systems: Estimation, Identification and Adaptive Control*. Prentice-Hall, 1986. 37

[KvD87] J.J. Koenderink and A.J. van Doorn. Representation of local geometry in the visual system. *Biological Cybernetics*, 55:367–375, 1987. 47, 60

[KYK01] H. Kimura, T. Yamashita, and S. Kobayashi. Reinforcement learning of walking behavior for a four-legged robot. In *Proc. of the 40th IEEE Conference on Decision and Control*, Orlando (FL, USA), December 2001. 38

[KZB04] T. Kadir, A. Zisserman, and M. Brady. An affine invariant salient region detector. In *Proc. of the 8th European Conference on Computer Vision*, pages 228–241, Prague (Czech Republic), May 2004. 58

[LF06] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006. 65

[Lin91] L.-J. Lin. Programming robots using reinforcement learning and teaching. In *Proc. of the 9th National Conference on Artificial Intelligence*, pages 781–786, Cambridge (MA, USA), 1991. 126, 127

[Lin93] L.J. Lin. *Reinforcement Learning for Robots using Neural Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh, 1993. 37

[Lin98] T. Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):77–116, 1998. 54, 57

[Lip75] S.A. Lippman. Applying a new device in the optimization of exponential queueing systems. *Operation Research*, 23:687–710, 1975. 21

[LMB$^+$05] D.A. Lisin, M.A. Mattar, M.B. Blaschko, M.C. Benfield, and E.G. Learned-Miller. Combining local and global image features for object class recognition. In *Proc. of the IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, June 2005. 42

[LMGY04] T. Liu, A.W. Moore, A.G. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems*, Vancouver (British Columbia, Canada), December 2004. 65

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, pages 1150–1157, Corfu (Greece), September 1999. xvii, 56, 62

[Low04] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. xvii, 10, 62, 105, 162

[Low05] D.G. Lowe. Implementation of SIFT, 2005. http://www.cs.ubc.ca/~lowe/keypoints/. 63

[LP03] M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003. xvii, 13, 38, 121, 129, 130, 131

[LS03] B. Leibe and B. Schiele. Analyzing appearance and contour based methods for object categorization. In *Proc. of the Conference on Computer Vision and Pattern Recognition*, pages 409–415, Madison (WI, USA), June 2003. IEEE Computer Society. 46

[Luc59] R.D. Luce. *Individual Choice Behavior*. Wiley, New York, 1959. 34

[MA93] A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993. 38

[MA95] A. Moore and C. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21, 1995. 76, 115, 116

[Man60] A.S. Manne. Linear programming and sequential decisions. *Management Science*, 6:259–267, 1960. 24

[Mar82a] D. Marr. *Vision*. Freeman, San Francisco, 1982. 2

[Mar82b] D. Marr. *Vision*. Freeman, 1982. 56

[Mar05] R. Marée. *Classification Automatique d'Images par Arbres de Décision*. PhD thesis, University of Liège, Liège (Belgium), February 2005. 65, 136, 172

[MBM99] R. Munos, L. Baird, and A. Moore. Gradient descent approaches to neural-net-based solutions of the hamilton-jacobi-bellman equation. In *International Joint Conference on Neural Networks*, page 6, July 1999. 31

[MCA$^+$01] M.E. McCarty, R.K. Clifton, D.H. Ashmead, P. Lee, and N. Goubet. How infants use vision for grasping objects. *Child Development*, 72:973–987, 2001. 3

[McC96] R.A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, New York, 1996. 10, 75, 79, 105, 168, 171

[MCUP04] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767, 2004. 58

[MD89] J. Moody and C.J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989. 124

[MGD98] P. Montesinos, V. Gouet, and R. Deriche. Differential invariants for color images. In *Proc. of the International Conference on Pattern Recognition*, pages 838–840, Brisbane (Australia), August 1998. 61

[MGPW05] R. Marée, P. Geurts, J. Piater, and L. Wehenkel. Random subwindows for robust image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 34–40, San Diego (CA, USA), June 2005. 43, 52, 54, 59, 63, 65, 136, 138, 143, 145, 172

[Mik06] K. Mikolajczyk. Implementation of various detectors and descriptors, 2006. http://www.robots.ox.ac.uk/~vgg/research/affine/. 63

[Mit97] T.M. Mitchell. *Machine Learning*. McGraw Hill, 1997. 123

[MM96] B.S. Manjunath and W.Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):837–842, 1996. 61

[MM02] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49:291–323, December 2002. 69, 76, 77, 149, 172

[MMD05] T. Martínez-Marín and T. Duckett. Fast reinforcement learning for vision-guided mobile robots. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 18–22, Barcelona (Spain), April 2005. 8

[MN95] H. Murase and S.K. Nayar. Visual learning and recognition of 3-D objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995. 45

[Mor80] H. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical Report CMU-RI-TR-3, Carnegie-Mellon University, September 1980. 47, 48

[MS01a] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *Proc. of the 8th International Conference on Computer Vision*, pages 525–531, Vancouver (Canada), July 2001. 56

[MS01b] J. Moody and M. Saffell. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889, 2001. 38

[MS02] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *Proc. of the 7th European Conference on Computer Vision*, volume 1, pages 128–142, 2002. 48, 57

[MS03] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 257–263, Madison (WI, USA), June 2003. 10, 62

[MS04] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, 2004. 48, 53, 56, 57, 58

[MS05] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005. 10, 47, 61, 62

[MS06] R. Munos and C. Szepesvári. Finite time bounds for sampling based fitted value iteration, 2006. Submitted to Journal of Machine Learning Research. 127, 128

[MSN05] J. Michels, A. Saxena, and A.Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proc. of the 22nd International Conference in Machine Learning*, pages 593–600, Bonn (Germany), August 2005. 8

[MTS+05] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(7):43–72, November 2005. 10, 57, 58

[Mun03] R. Munos. Error bounds for approximate policy iteration. In *International Conference on Machine Learning*, pages 560–567, 2003. 121, 129

[Mun05] R. Munos. Error bounds for approximate value iteration. In *Proc. of the 20th National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pages 1006–1011, Pittsburgh (Pennsylvania, USA), July 2005. 128

[Mun06a] R. Munos. Performance bounds for approximate value iteration, 2006. Submitted to SIAM Journal on Control and Optimization. 128, 132

[Mun06b] R. Munos. Policy gradient in continuous time. *Journal of Machine Learning Research*, 7:771–791, 2006. 31

[Mur02] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, July 2002. 73

[MWSP04] C.K. Monson, D. Wingate, K.D. Seppi, and T.S. Peterson. Variable resolution discretization in the joint space. In *International Conference on Machine Learning and Applications*, 2004. 149, 163

[NB03] A. Nedić and D.P. Bertsekas. Least-squares policy evaluation algorithms with linear function approximation. *Journal of Discrete Event Systems*, 13:79–110, 2003. 127

[NBE+93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E.H. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture and shape. In *Storage and Retrieval for Image and Video Databases*, pages 173–187, San Jose (CA, USA), February 1993. 46

[NCD+04] A.Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, Ben Tse, B. Berger, and E. Liang. Inverted autonomous helicopter flight via reinforcement learning. In *Proc. of the International Symposium on Experimental Robotics*, 2004. 38

[NM65] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965. 149, 172

[NMN94] S.K. Nayar, H. Murase, and S.A. Nene. Learning, positioning, and tracking visual appearance. In *Proc. of the Internation Conference on Robotics and Automation*, San Diego (CA, USA), May 1994. 45

[NNM96] S.A. Nene, S.K. Nayar, and H. Murase. Columbia object image library (COIL-100). Technical Report CUCS-006-96, Columbia University, New York, February 1996. 89, 92, 116

[OD04] G.C. Oana and K.J. Dana. 3D texture recognition using bidirectional feature histograms. *International Journal of Computer Vision*, 59(1):33–60, 2004. 46

[ODS+04] J. O'Doherty, P. Dayan, J. Schultz, R. Deichmann, K. Friston, and R. Dolan. Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science*, 304:452–454, April 2004. 39

[OG06] OurGrid, 2006.
http://www.ourgrid.org/. 141

[OI97] K. Ohba and K. Ikeuchi. Detectability, uniqueness, and reliability of eigen windows for stable verification of partially occluded objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1043–1048, 1997. 59

[O'L92] D.D. O'Leary. Development of connectional diversity and specificity in the mammalian brain by the pruning of collateral projections. *Current Opinion in Neurobiology*, 2(1):70–77, February 1992. 3

[OM02] Š. Obdržálek and J. Matas. Local affine frames for image retrieval. In *Proc. of the International Conference on Image and Video Retrieval*, pages 318–327, 2002. 62, 64

[OM03] Š. Obdržálek and J. Matas. Image retrieval using local compact DCT-based representation. In *Proc. of the 25th DAGM Symposium*, pages 490–497, Magdeburg (Germany), 2003. 62

[OM05] Š. Obdržálek and J. Matas. Sub-linear indexing for large scale object recognition. In *Proc. of the 16th British Machine Vision Conference*, volume 1, pages 1–10, September 2005. 65

[OS02] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine learning*, 49(2–3):161–178, 2002. 128

[PB98] J. Peng and B. Bhanu. Closed-loop object recognition using reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(2):139–154, February 1998. 8

[PC05] J.M. Porta and E. Celaya. Reinforcement learning for agents with many sensors and actuators acting in categorizable environments. *Journal of Artificial Intelligence Research*, 23:79–122, 2005. 69, 75, 149, 151

[PEL00] J.C. Principe, N.R. Euliano, and W.C. Lefebvre. *Neural and Adaptive Systems: Fundamentals Through Simulations*. John Wiley & Sons, 2000. 125

[Pes02] L. Peshkin. *Reinforcement Learning with Policy Search*. PhD thesis, Brown University, 2002. 128

[PFS05] L. Paletta, G. Fritz, and C. Seifert. Q-learning of sequential attention for visual object recognition from informative local descriptors. In *Proc. of the 22nd International Conference on Machine Learning (ICML)*, pages 649–656, Bonn (Germany), August 2005. 8, 10

[PFZ03] P. Perona, R. Fergus, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Conference on Computer Vision and Pattern Recognition*, volume 2, page 264, June 2003. 111

[PG02] J.H. Piater and R.A. Grupen. Learning appearance features to support robotic manipulation. In *Cognitive Vision Workshop*, September 2002. 9

[PH01] L.D. Pyeatt and A.E. Howe. Decision tree function approximation in reinforcement learning. In *Proc. of the Third International Symposium on Adaptive Systems*, pages 70–77, Havana, Cuba, March 2001. 76, 78, 84

[Pia01] J.H. Piater. *Visual Feature Learning*. PhD thesis, University of Massachusetts, Computer Science Department, Amherst (MA, USA), February 2001. 6, 9, 110, 120, 121, 167

[POP98] C.P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *Proc. of the 6th International Conference on Computer Vision*, pages 555–562, January 1998. 61

[Pow87] M.J.D. Powell. Radial basis functions for multivariate interpolation: A review. In J.C. Mason and M.G. Cox, editors, *Algorithms for Approximation*, pages 143–167. Clarendon Press, 1987. 124

[PP93] N.R. Pal and S.K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277–1294, September 1993. 44

[PPJV01] R. Paredes, J.C. Pérez-Cortes, A. Juan, and E. Vidal. Local representations and a direct voting scheme for face recognition. In *Proc. of the 1st International Workshop on Pattern Recognition in Information Systems*, pages 71–79, July 2001. 59

[PRB05] L. Paletta, E. Rome, and H. Buxton. Attention architectures for machine vision and mobile robots. In L. Itti, G. Rees, and J.K. Tsotsos, editors, *Neurobiology of Attention*, pages 642–648. Academic Press/Elsevier, 2005. 10

[PS78] M.L. Puterman and M.C. Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24:1127–1137, 1978. 27, 28

[PV98] M. Pontil and A. Verri. Support vector machines for 3D object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:637–646, 1998. 45

[PW93] J. Peng and R.J. Williams. Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993. 38

[PW96] J. Peng and R.J. Williams. Incremental multi-step Q-learning. *Machine Learning*, 22:283–290, 1996. 36

[PW03] B. Porr and F. Wörgötter. Isotropic sequence order learning. *Neural Computation*, 15(4):831–864, 2003. 3

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco (CA, USA), 1993. 84

[RA98] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proc. of the 15th International Conference on Machine Learning*, pages 463–471, Madison (WI, USA), 1998. Morgan Kaufmann. 38

[RH99] T. Randen and J. Husøy. Filter for texture classification: A comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):291–310, 1999. 61

[Rie05] M. Riedmiller. Neural reinforcement learning to swing-up and balance a real pole. In *Proc. of the International Conference on Systems, Man and Cybernetics*, Big Island (USA), October 2005. 128

[RM51] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951. 33

[RM86] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, volume 1. MIT Press, Cambridge (MA, USA), 1986. 124

[RM04] L.W. Renninger and J. Malik. When is scene identification just texture recognition? *Vision Research*, 44(19):2301–2311, September 2004. 46

[RN94] G.A. Rummery and M. Niranjan. On-line Q-learning using connectionist sytems. Technical Report CUED/F-INFENG-TR 166, Cambridge University, 1994. 35, 37

[RP03] B. Ratitch and D. Precup. Using MDP characteristics to guide exploration in reinforcement learning. In *Proc. of the 4th European Conference on Machine Learning*, pages 313–324, Dubrovnik (Croatia), September 2003. 34

[RPTB01] Y. Rubner, J. Puzicha, C. Tomasi, and J.M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. *Computer Vision and Image Understanding*, 84(1):25–43, October 2001. 64

[RTG00] Y. Rubner, C. Tomasi, and L.J. Guibas. The Earth Mover's Distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000. 64

[Rub81] R.Y. Rubinstein. *Simulation and the Monte-Carlo Method.* Wiley, New-York, 1981. 37

[Rum95] G.A. Rummery. *Problem Solving with Reinforcement Learning.* PhD thesis, Cambridge University, 1995. 37, 149

[Sal93] M. Salganicoff. Density-adaptive learning and forgetting. In *Proc. of the 10th International Conference on Machine Learning*, pages 276–283, Amherst (MA, USA), June 1993. Morgan Kaufmann Publishers. 76

[Sal02] B. Sallans. *Reinforcement Learning for Factored Markov Decision Processes.* PhD thesis, University of Toronto, 2002. 73

[Sam84] H. Samet. The Quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984. 65, 125

[SB90] R.S. Sutton and A.G. Barto. Time-derivative models of pavlovian reinforcement. In M. Gabriel and J. Moore, editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pages 497–537. MIT Press, 1990. 7

[SB91] M.J. Swain and D.H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991. 45, 64

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement Learning, an Introduction*. MIT Press, 1998. xvii, 3, 6, 15, 16, 24, 26, 28, 29, 30, 35, 36, 37, 38

[SC95] J.R. Smith and S-F. Chang. Single color extraction and image query. In *IEEE International Conference on Image Processing*, pages 528–531, Washington, DC, October 1995. 46

[SC96] B. Schiele and J.L. Crowley. Object recognition using multidimensional receptive field histograms. In *Proc. of the 4th European Conference on Computer Vision*, Cambridge (UK), April 1996. 46, 64

[SC00] B. Schiele and J.L. Crowley. Recognition without correspondence using multidimensional receptive field histograms. *International Journal of Computer Vision*, 36(1):31–50, January 2000. 46

[Sca04] F. Scalzo. Unsupervised learning of visual feature hierarchies, 2004. DEA Thesis, University of Liège. 55

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986. 99

[Sch91] J. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Institut für Informatik, Technische Universität at München (Germany), 1991. 35

[Sch97] S. Schaal. Learning from demonstration. In M. C. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 1040–1046. Cambridge, MA, MIT Press, 1997. 8

[Sch03] R. Schoknecht. Optimality of reinforcement learning algorithms with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 1555–1562. MIT Press, Cambridge (MA, USA), 2003. 127

[Ser79] R.F. Serfozo. An equivalence between continuous and discrete time Markov decision processes. *Operation Research*, 27:616–620, 1979. 21

[Ser82] J. Serra. *Image analysis and mathematical morphology*. Academic Press, New York, 1982. 43

[SG99] C. Stauffer and W.E.L. Grimson. Adaptive background mixture models for real-time tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 246–252, Fort Collins (CO, USA), 1999. 44

[Sha49] C.E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949. 100

[Sha53] L.S. Shapley. Stochastic games. In *Proc. of the National Academy of Sciences of the United States of America*, volume 39, pages 1095–1100, 1953. 16

[SI03] K. Shibata and M. Iida. Acquisition of box pushing by direct-vision-based reinforcement learning. In *Proc. of the Society of Instrument and Control Engineers Annual Conference*, page 6, 2003. 8, 11

[SIFW03] E.B. Sudderth, A.T. Ihler, W.T. Freeman, and A.S Willsky. Nonparametric belief propagation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 605–612, 2003. 111, 173

[SJJ95] S.P. Singh, T. Jaakkola, and M.I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems*, volume 7, pages 361–368. MIT Press, 1995. 81

[SJLS00] S.P. Singh, T. Jaakkola, M.L. Littman, and C. Szepesvari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000. 35

[SK87] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America A*, 4(3):519–524, 1987. 45

[SLDV98] P. Simard, Y. LeCun, J.S. Denker, and B. Victorri. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade (outgrowth of a 1996 NIPS workshop)*, volume 1524 of *Lecture Notes in Computer Science*, pages 239–274, London, UK, 1998. Springer-Verlag. 64

[SM97] C. Schmid and R. Mohr. Local greyvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–535, 1997. 10, 64, 65, 89, 111

[SM00] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. 44

[SM05] C. Szepesvári and R. Munos. Finite time bounds for sampling based fitted value iteration. In *Proc. of the 22nd International Conference on Machine Learning*, pages 880–887, Bonn (Germany), August 2005. 127, 128

[Sma02] W.D. Smart. *Making Reinforcement Learning Work on Real Robots*. PhD thesis, Brown University, Providence, 2002. 125, 126, 149

[SMB00] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000. 10, 50, 52, 54, 58

[SNJM04] S. Se, H. Ng, P. Jasiobedzki, and T. Moyung. Vision based modeling and localization for planetary exploration rovers. In *Proc. of the International Astronautical Congress*, 2004. 63

[SP05] F. Scalzo and J.H. Piater. Statistical learning of visual feature hierarchies. In *Proc. of the IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, San Diego (CA, USA), June 2005. IEEE. 13, 111

[SP06] F. Scalzo and J.H. Piater. Unsupervised learning of dense hierarchical appearance representations. In *Proc. of the 18th International Conference on Pattern Recognition*, Hong-Kong, August 2006. 13, 55, 59, 111, 120, 173

[SPI06] Spin (model checking software), 2006. `http://spinroot.com/spin/whatispin.html`. 99

[SR97] P.G. Schyns and L. Rodet. Categorization creates functional features. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 23(3):681–696, 1997. 3

[SS96] Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123, 1996. 37

[SS97] J.W. Sheppard and S. Salzberg. A teaching strategy for memory-based control. *Artificial Intelligence Review*, 11(1–5):343–370, 1997. 126

[SS01] P. Stone and R.S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proc. of the 18th International Conference on Machine Learning*, pages 537–544. Morgan Kaufmann, 2001. 38

[SSH06] Secure shell. Wikipedia, the Free Encyclopedia, 2006. `http://en.wikipedia.org/wiki/Ssh`. 141

[SSR98] J.C. Santamaria, R.S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6(2):163–218, 1998. 148

[SSTVG03] H. Shao, T. Svoboda, T. Tuytelaars, and L. Van Gool. HPAT indexing for fast object/scene recognition based on local appearance. In *Proc. of the 2nd Internation Conference on Image and Video Retrieval*, pages 71–80, Urbana-Champaign (IL, USA), July 2003. 62, 64

[STLC97] S. Sclaroff, L. Taycher, and M. La Cascia. ImageRover: A content-based image browser for the world wide web. *IEEE Workshop on Content-Based Access Image and Video Libraries*, page 2, 1997. 46

[Sut88] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988. xvii, 32, 35, 37

[Sut90] R.S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proc. of the 7th International Conference on Machine Learning*, pages 216–224, San Mateo (CA, USA), 1990. Morgan Kaufmann. 38, 85, 89

[Sut96a] R. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press, 1996. 126

[Sut96b] R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proc. of Advances in Neural Information Processing Systems*, pages 1038–1044, Cambridge, MA, 1996. MIT Press. 35

[Sut04] R.S. Sutton. Reinforcement learning FAQ, 2004. `http://www.cs.ualberta.ca/~sutton/RL-FAQ.html`. 7, 24, 125

[SW96] D.L. Swets and J. Weng. Using discriminant eigenfeatures for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):831–836, August 1996. 45

[TC03] M.J. Tarr and Y.D. Cheng. Learning to see faces and objects. *Trends in Cognitive Sciences*, 7(1):23–30, 2003. 3, 4

[TC04] J. Thureson and S. Carlsson. Appearance based qualitative image description for object class recognition. In *Proc. of the 8th European Conference on Computer Vision*, pages 518–529, Prague (Czech Republic), May 2004. 62

[Tes95] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995. 38, 127

[tHK00] S. ten Hagen and B. Kröse. q-learning for systems with continuous state and action spaces. In *Proc. of the 10th Belgian-Dutch Conference on Machine Learning*, 2000. 126, 148

[Thr92] S. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1992. 35

[Tij94] H.C. Tijms. *Stochastic Models: An Algorithmic Approach*. John Wiley, 1994. 21

[TP91] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991. 45

[Tsi94] J.N. Tsitsiklis. Asynchronous stochastic approximation and *Q*-learning. *Machine Learning*, 16(3):185–202, 1994. 33

[Tso94] J.K. Tsotsos. There is no one way to look at vision. *CVGIP: Image Understanding*, 60(1):95–97, 1994. 5

[TTA99] Y. Takahashi, M. Takeda, and M. Asada. Continuous valued *Q*-learning for vision-guided behavior acquisition. In *Proc. of the International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 255–260, 1999. 8

[TV96] J.N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996. 126, 127

[TV97] J. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997. 127

[TVG04] T. Tuytelaars and L. Van Gool. Matching widely separated views based on affine invariant regions. *International Journal of Computer Vision*, 59(1):61–85, 2004. 57, 58

[Uth02] W.T.B. Uther. *Tree Based Hierarchical Reinforcement Learning*. PhD thesis, Carnegie Mellon University, 2002. 76

[UV98] W.T.B. Uther and M.M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proc. of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 769–774, Madison (WI, USA), July 1998. 76, 79

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518, Kauai (HI, USA), December 2001. 58

[Vou06] L. Voulzy. Derniers baisers. In *Septième Vague*, June 2006. http://www.montefiore.ulg.ac.be/~jodogne/private/writing-phd.jpg.

[VZ03] M. Varma and A. Zisserman. Texture classification: Are filter banks necessary? In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 691–698, Madison (WI, USA), June 2003. 61

[Wal47] A. Wald. *Sequential Analysis.* John Wiley, 1947. 16

[Wal50] A. Wald. *Statistical Decision Functions.* John Wiley, 1950. 16

[Wat89] C.J.C.H. Watkins. *Learning From Delayed Rewards.* PhD thesis, King's College, Cambridge (UK), 1989. 24, 32, 34, 36, 82, 126

[WB91] S.D. Whitehead and D.H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–83, 1991. 72, 74

[WB99] G. Wallis and H. Bülthoff. Learning to recognize objects. *Trends In Cognitive Sciences*, 3(1):22–31, January 1999. 111

[WC96] J. Weng and S. Chen. Incremental learning for vision-based navigation. In *Proc. of the International Conference on Pattern Recognition*, pages 45–49, Vienna (Austria), 1996. 45

[WD92] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992. 24, 33

[Wer90] P.J. Werbos. Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, 3:179–189, 1990. 126

[Wes95] C.-J. Westelius. *Focus of Attention and Gaze Control for Robot Vision.* PhD thesis, Linköping University (Sweden), 1995. 5

[WGZ99] D. Wettergreen, C. Gaskett, and A. Zelinsky. Autonomous guidance and control for an underwater robotic vehicle. In *Proc. of the International Conference on Field and Service Robotics*, Pittsburgh (USA), August 1999. 8

[Whi91] S.D. Whitehead. Complexity and cooperation in q-learning. In *Proc. of the 8th International Workshop on Machine Learning*, pages 363–367, Evanston, IL, 1991. 37

[Whi93] D.J. White. *Markov Decision Processes.* John Wiley, New York, 1993. 21, 22

[Wie99] M.A. Wiering. *Explorations in Efficient Reinforcement Learning.* PhD thesis, University of Amsterdam (IDSIA), February 1999. 35

[Wie05] M. Wiering. $QV(\lambda)$-learning: A new on-policy reinforcement learning algorithm. In *Proc. of the 7th European Workshop on Reinforcement Learning*, pages 17–18, Napoli (Italy), October 2005. 37

[WK02] H. Wersing and E. Körner. Unsupervised learning of combination features for hierarchical recognition models. In *Proc. of the International Conference on Artificial Neural Networks*, volume 2415 of *Lecture Notes in Computer Science*, pages 1225–1230. Springer, August 2002. 111

[WS98] M.A. Wiering and J. Schmidhuber. Fast online $Q(\lambda)$. *Machine Learning*, 33(1):105–116, 1998. 37

[WSV99] N. Winters and J. Santos-Victor. Omni-directional visual navigation. In *Proc. of the 7th International Symposium on Intelligent Robotic Systems*, pages 109–118, Coimbra (Portugal), July 1999. 45

[WWZ04] C. Weber, S. Wermter, and A. Zochios. Robot docking with neural vision and reinforcement. *Knowledge-Based Systems*, 17(2–4):165–172, 2004. 8

[Yin02] Peng-Yeng Yin. Maximum entropy-based optimal threshold selection using deterministic reinforcement learning with controlled randomization. *Signal Processing*, 82:993–1006, 2002. 8

[YIS99] J. Yoshimoto, S. Ishii, and M. Sato. Application of reinforcement learning to balancing ACROBOT. In *Proc. of the 1999 IEEE International Conference on Systems, Man and Cybernetics*, pages 516–521, 1999. 38

[You87] R.A. Young. The Gaussian derivative model for spatial vision: I. Retinal mechanisms. *Spatial Vision*, 2(4):273–293, 1987. 47

[ZD95] W. Zhang and T.G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, pages 1114–1120, San Francisco (CA, USA), 1995. 127

[ZMLS06] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. In *Proc. of Beyond Patches Workshop, in conjunction with the IEEE Conference on Computer Vision and Pattern Recognition*, 2006. 65

APPENDIX

ONE

Proofs about Markov Decision Processes

Proof of Theorem 2.13 (page 20). We successively obtain:

$$
\begin{aligned}
V^{\pi}(s) &= \mathrm{E}^{\pi}\left\{\sum_{t=0}^{\infty}\gamma^{t}\mathcal{R}(s_t,a_t)\mid s_0=s\right\}\\
&= \mathrm{E}^{\pi}\left\{\mathcal{R}(s_0,a_0)+\gamma\sum_{t=0}^{\infty}\gamma^{t}\mathcal{R}(s_{t+1},a_{t+1})\mid s_0=s\right\}\\
&= \sum_{a\in A}\pi(s,a)\ \mathrm{E}^{\pi}\left\{\mathcal{R}(s_0,a_0)+\gamma\sum_{t=0}^{\infty}\gamma^{t}\mathcal{R}(s_{t+1},a_{t+1})\mid s_0=s,a_0=a\right\}\\
&= \sum_{a\in A}\pi(s,a)\left[\mathcal{R}(s_0,a)+\gamma\ \mathrm{E}^{\pi}\left\{\sum_{t=0}^{\infty}\gamma^{t}\mathcal{R}(s_{t+1},a_{t+1})\mid s_0=s,a_0=a\right\}\right]\\
&= \sum_{a\in A}\pi(s,a)\left[\mathcal{R}(s_0,a)+\gamma\sum_{s'\in S}\mathcal{T}(s,a,s')\ \mathrm{E}^{\pi}\left\{\sum_{t=0}^{\infty}\gamma^{t}\mathcal{R}(s_{t+1},a_{t+1})\mid s_1=s'\right\}\right]\\
&= \sum_{a\in A}\pi(s,a)\left[\mathcal{R}(s_0,a)+\gamma\sum_{s'\in S}\mathcal{T}(s,a,s')\ V^{\pi}(s')\right],
\end{aligned}
$$

which concludes the proof. □

Lemma A.1. Let $f,f':B\mapsto\mathbb{R}$ be two functions that map a finite set I to the real numbers. Then:

$$\left|\max_{i\in I}f(i)-\max_{i\in I}f'(i)\right|\le\max_{i\in I}|f(i)-f'(i)|. \tag{A.1}$$

Proof. This theorem can be seen as a consequence of triangle inequality. More formally, let $k,k'\in I$ be respectively such that $f(k)=\max_{i\in I}f(i)$ and $f'(k')=\max_{i\in I}f'(i)$. We can assume without loss of generality that $f(k)\ge f'(k')$ (otherwise, exchange f and f'). As a consequence and by definition of k', $f(k) \geq f'(k)$. Also note that $f'(k') \geq f'(k)$. Therefore:

$$\begin{aligned}\left|\max_{i\in I} f(i) - \max_{i\in I} f'(i)\right| &= |f(k) - f'(k')| = f(k) - f'(k') \\ &\leq f(k) - f'(k) = |f(k) - f'(k)| \\ &\leq \max_{i\in I} |f(i) - f'(i)|.\end{aligned}$$

□

Proof of Theorem 2.19 (page 23). We directly prove the two relations $||T^\pi V - T^\pi V'||_\infty \leq \gamma ||V - V'||_\infty$ and $||TV - TV'||_\infty \leq \gamma ||V - V'||_\infty$. Note that in both case, the contraction factor ρ equals γ. For T^π, we successively obtain:

$$\begin{aligned}||T^\pi V - T^\pi V'||_\infty &= \max_{s\in S} \left| \sum_{a\in A} \pi(s,a) \left(\mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s')V(s') \right) - \right. \\ &\qquad \left. \sum_{a\in A} \pi(s,a) \left(\mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s')V'(s') \right) \right| \\ &= \max_{s\in S} \left| \sum_{a\in A} \pi(s,a) \left(\gamma \sum_{s'\in S} \mathcal{T}(s,a,s')(V(s') - V'(s')) \right) \right| \\ &= \gamma \max_{s\in S} \sum_{a\in A} \pi(s,a) \sum_{s'\in S} \mathcal{T}(s,a,s')|V(s') - V'(s')| \\ &\leq \gamma \max_{s\in S} \sum_{a\in A} \pi(s,a) \sum_{s'\in S} \mathcal{T}(s,a,s')||V - V'||_\infty \\ &= \gamma ||V - V'||_\infty \max_{s\in S} \sum_{a\in A} \pi(s,a) \sum_{s'\in S} \mathcal{T}(s,a,s') \\ &= \gamma ||V - V'||_\infty,\end{aligned}$$

where the last equality follows from the fact that probability distributions sum to one. Then, we get for T:

$$\begin{aligned}||TV - TV'||_\infty &= \max_{s\in S} \left| \max_{a\in A} \left(\mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s')V(s') \right) - \right. \\ &\qquad \left. \max_{a\in A} \left(\mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s')V'(s') \right) \right|\end{aligned}$$

$$\begin{aligned}
&\leq \max_{s\in S}\max_{a\in A}\Bigg|\mathcal{R}(s,a)+\gamma\sum_{s'\in S}\mathcal{T}(s,a,s')V(s')\, - \\
&\qquad\qquad \mathcal{R}(s,a)-\gamma\sum_{s'\in S}\mathcal{T}(s,a,s')V'(s')\Bigg| \\
&= \gamma\max_{s\in S}\max_{a\in A}\left|\sum_{s'\in S}\mathcal{T}(s,a,s')(V(s')-V'(s'))\right| \\
&= \gamma\max_{s\in S}\max_{a\in A}\sum_{s'\in S}\mathcal{T}(s,a,s')|V(s')-V'(s')| \\
&= \gamma\max_{s\in S}\max_{a\in A}\sum_{s'\in S}\mathcal{T}(s,a,s')||V-V'||_\infty \\
&\leq \gamma||V-V'||_\infty\max_{s\in S}\max_{a\in A}\sum_{s'\in S}\mathcal{T}(s,a,s') \\
&= \gamma||V-V'||_\infty,
\end{aligned}$$

where the first inequality is a consequence of Lemma A.1. □

Lemma A.2. A Markovian, stationary control policy π is optimal if and only if $T^\pi V^* = V^*$.

Proof. Suppose that π is optimal. Bellman optimality theorem shows that $V^\pi = V^*$. Furthermore, Theorem 2.13 states that $T^\pi V^\pi = V^\pi$. Hence, $T^\pi V^* = T^\pi V^\pi = V^\pi = V^*$. Conversely, suppose that $T^\pi V^* = V^*$. By Banach fixed point theorem, V^* is the unique fixed point of T^π. Then, as a consequence of Theorem 2.13, we get $V^\pi = V^*$. Bellman optimality theorem concludes that π is optimal. □

Proof of Theorem 2.21 (page 23). According to Lemma A.2, it is sufficient to prove that the policy π^* defined by Equation 2.17 satisfies $T^{\pi^*}V^* = V^*$:

$$\begin{aligned}
T^{\pi^*}V^*(s) &= \mathcal{R}(s,\pi^*(s))+\gamma\sum_{s'\in S}\mathcal{T}(s,\pi^*(s),s')V^*(s') \\
&= \max_{a\in A}\left(\mathcal{R}(s,a)+\gamma\sum_{s'\in S}\mathcal{T}(s,a,s')V^*(s')\right) \\
&= V^*(s),
\end{aligned}$$

for all $s \in S$. The first line results from Corollary 2.14, the second line from the definition of π^*, and the third line from Bellman optimality theorem. □

Theorem A.3. *H is a contraction mapping with respect to the maximum norm* $||Q||_\infty = \max_{(s,a)\in S\times A}|Q(s,a)|$.

Proof. The proof is mostly identical to that of Theorem 2.19. Let Q and Q' be two state-action value functions. We have for H^π:

$$\begin{aligned}
& || \; H^\pi Q - H^\pi Q' ||_\infty \\
= \; & \max_{s\in S} \max_{a\in A} \left| \mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') \sum_{a'\in A} \pi(s',a') Q(s',a') - \right. \\
& \left. \mathcal{R}(s,a) - \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') \sum_{a'\in A} \pi(s',a') Q'(s',a') \right| \\
= \; & \max_{s\in S} \max_{a\in A} \left| \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') \sum_{a'\in A} \pi(s',a') (Q(s',a') - Q'(s',a')) \right| \\
= \; & \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} \mathcal{T}(s,a,s') \sum_{a'\in A} \pi(s',a') |Q(s',a') - Q'(s',a')| \\
\leq \; & \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} \mathcal{T}(s,a,s') \sum_{a'\in A} \pi(s',a') ||Q - Q'||_\infty \\
= \; & \gamma ||Q - Q'||_\infty \; \max_{s\in S} \max_{a\in A} \sum_{s'\in S} \mathcal{T}(s,a,s') \sum_{a'\in A} \pi(s',a') \\
= \; & \gamma ||Q - Q'||_\infty .
\end{aligned}$$

□

Theorem A.4. H^π is a contraction mapping with respect to the maximum norm $||Q||_\infty = \max_{(s,a)\in S\times A} |Q(s,a)|$.

Proof. Once again, the proof is mostly identical to that of Theorem 2.19. Let Q and Q' be two state-action value functions. We have:

$$\begin{aligned}
|| \; HQ - HQ' ||_\infty \; = \; & \max_{s\in S} \max_{a\in A} \left| \mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') \max_{a'\in A} Q(s',a') - \right. \\
& \left. \mathcal{R}(s,a) - \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') \max_{a'\in A} Q'(s',a') \right| \\
= \; & \max_{s\in S} \max_{a\in A} \left| \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') \left(\max_{a'\in A} Q(s',a') - \max_{a'\in A} Q'(s',a') \right) \right| \\
= \; & \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} \mathcal{T}(s,a,s') \left| \max_{a'\in A} Q(s',a') - \max_{a'\in A} Q'(s',a') \right| \\
\leq \; & \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} \mathcal{T}(s,a,s') \max_{a'\in A} |Q(s',a') - Q'(s',a')| \\
\leq \; & \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} \mathcal{T}(s,a,s') ||Q - Q'||_\infty
\end{aligned}$$

$$\begin{aligned} &= \gamma ||Q - Q'||_\infty \max_{s \in S} \max_{a \in A} \sum_{s' \in S} \mathcal{T}(s, a, s') \\ &= \gamma ||Q - Q'||_\infty . \end{aligned}$$

□

Definition A.5. A square matrix $A \in \mathbb{R}^{n \times n}$ is a *stochastic matrix* if:

1. $A_{ij} \geq 0$ for all $i, j = 1, \ldots, n$;
2. $\sum_{j=1}^{n} A_{ij} = 1$ for all $i = 1, \ldots, n$.

Note that stochastic matrices are a particular subclass of nonnegative matrices. They play an important role in the analysis of Markov chains [BP79].

Lemma A.6. Let A and B be two stochastic matrix of same dimensions. Then their product AB is also a stochastic matrix.

Proof. This proof is quite elementary. The definition of the matrix product gives:

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}.$$

Thus, we directly conclude that $(AB)_{ij}$ is a matrix of nonnegative elements, by definition of A and B. Furthermore, for all $i = 1, \ldots, n$:

$$\sum_{j=1}^{n} (AB)_{ij} = \sum_{j=1}^{n} \sum_{k=1}^{n} A_{ik} B_{kj} = \sum_{k=1}^{n} A_{ik} \sum_{j=1}^{n} B_{kj} = \sum_{k=1}^{n} A_{ik} = 1,$$

which concludes the proof. □

Lemma A.7. Let A be a square matrix. If

$$\lim_{n \to \infty} A^n = \mathbf{0}, \tag{A.2}$$

where $\mathbf{0}$ is the zero matrix, then $(I - A)$ has an inverse, and

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k. \tag{A.3}$$

Proof. This proof is essentially taken from Kemeny and Snell [KS60, Theorem 1.11.1]. Consider the identity:

$$\begin{aligned} I - A^n &= \left(\sum_{k=0}^{n-1} A^k\right) - \left(\sum_{k=1}^{n} A^k\right) \\ &= \left(\sum_{k=0}^{n-1} A^k\right) - A\left(\sum_{k=0}^{n-1} A^k\right) \\ &= (I - A)\sum_{k=0}^{n-1} A^k, \end{aligned} \tag{A.4}$$

which holds for any $n > 0$. The hypothesis on A gives:

$$\lim_{n\to\infty}(I - A^n) = I - \lim_{n\to\infty} A^n = I.$$

Thus, for sufficiently large n, $I - A^n$ must have a non-zero determinant:

$$(\exists n') \det\left(I - A^{n'}\right) \neq 0.$$

This inequation along with Equation A.4 implies that:

$$\det\left(I - A^{n'}\right) = \det\left((I - A)\sum_{k=0}^{n'-1} A^k\right) = \det(I - A)\det\left(\sum_{k=0}^{n'-1} A^k\right) \neq 0.$$

As a consequence, $\det(I - A) \neq 0$ and the matrix $(I - A)$ has an inverse. Since this inverse exist, we can multiply both sides of Equation A.4 by it:

$$(I - A)^{-1}(I - A^n) = \sum_{k=0}^{n-1} A^k.$$

But the left side of this new identity clearly tends to $(I - A)^{-1}$ when n tends to infinity, which completes the proof. □

Proof of Theorem 2.24 (page 28). We first introduce a theoretical framework that allows us to reformulate the Policy Improvement theorem using matrix notation. Given the policy $\tilde{\pi}$, define the matrices $P^{\tilde{\pi}}$ and $R^{\tilde{\pi}}$ as follows:

$$P^{\tilde{\pi}}(s, s') = \sum_{a\in A} \tilde{\pi}(s, a)\mathcal{T}(s, a, s'), \tag{A.5}$$

$$R^{\tilde{\pi}}(s) = \sum_{a\in A} \tilde{\pi}(s, a)\mathcal{R}(s, a). \tag{A.6}$$

Note that $P^{\tilde{\pi}}(s', s)$ and $R^{\tilde{\pi}}(s)$ can be thought of as matrices, as S is assumed finite. $P^{\tilde{\pi}}(s, s')$ corresponds to the probability of reaching a state $s' \in S$ when starting from a state $s \in S$ and following the policy $\tilde{\pi}$. As for $R^{\tilde{\pi}}(s)$, it gives the expected immediate reward in a state $s \in S$ when following the policy $\tilde{\pi}$. We now make three observations:

1. Using this matrix notation, the Bellman equation 2.7 for $\tilde{\pi}$ becomes:

$$V^{\tilde{\pi}} = R^{\tilde{\pi}} + \gamma P^{\tilde{\pi}} V^{\tilde{\pi}}. \tag{A.7}$$

2. Secondly, we remark that $P^{\tilde{\pi}}$ is by definition a stochastic matrix (cf. Definition A.5). Therefore, by Lemma A.6, $(P^{\tilde{\pi}})^n$ is also a stochastic matrix for all $n \geq 0$. Because all the elements of a stochastic matrix are necessarily smaller than 1 and because $0 \leq \gamma < 1$, we get:

$$\lim_{n\to\infty} (\gamma P^{\tilde{\pi}})^n = \lim_{n\to\infty} \gamma^n \cdot \lim_{n\to\infty} (P^{\tilde{\pi}})^n \leq \lim_{n\to\infty} \gamma^n \mathbf{1} = \mathbf{0},$$

   where $\mathbf{1}$ is the matrix that contains only ones. Thus, Lemma A.7 can be applied on $A = \gamma P^{\tilde{\pi}}$. We deduce that $\gamma P^{\tilde{\pi}}$ admits an inverse and that:

$$(I - \gamma P^{\tilde{\pi}})^{-1} = \sum_{k=0}^{\infty} (\gamma P^{\tilde{\pi}})^k.$$

   Furthermore, because $\gamma \geq 0$ and because $(P^{\tilde{\pi}})^k$ are stochastic, hence nonnegative matrices for all $k \geq 0$, we conclude that $(I - \gamma P^{\tilde{\pi}})^{-1}$ is itself nonnegative.

3. Finally, the hypothesis about the relation between π and $\tilde{\pi}$ can be rewritten as:

$$\begin{aligned}
& \sum_{a\in A} \tilde{\pi}(s,a) Q^{\pi}(s,a) \geq V^{\pi}(s) \\
\Leftrightarrow\ & \sum_{a\in A} \tilde{\pi}(s,a) \left(\mathcal{R}(s,a) + \gamma \sum_{s'\in S} \mathcal{T}(s,a,s') V^{\pi}(s') \right) \geq V^{\pi}(s) \\
\Leftrightarrow\ & R^{\tilde{\pi}}(s) + \gamma \sum_{s'\in S} P^{\tilde{\pi}}(s,s') V^{\pi}(s') \geq V^{\pi}(s) \\
\Leftrightarrow\ & \left(R^{\tilde{\pi}} + \gamma P^{\tilde{\pi}} V^{\pi} - V^{\pi} \right)(s) \geq 0,
\end{aligned}$$

   for all $s \in S$, where the first equivalence is a consequence of Equation 2.20. Thus, $R^{\tilde{\pi}} + \gamma P^{\tilde{\pi}} V^{\pi} - V^{\pi}$ is also a nonnegative matrix.

Now consider the identity:

$$\begin{aligned}
& (I - \gamma P^{\tilde{\pi}})(V^{\tilde{\pi}} - V^{\pi}) = V^{\tilde{\pi}} - V^{\pi} - \gamma P^{\tilde{\pi}} V^{\tilde{\pi}} + \gamma P^{\tilde{\pi}} V^{\pi} \\
\Leftrightarrow\ & (I - \gamma P^{\tilde{\pi}})(V^{\tilde{\pi}} - V^{\pi}) = R^{\tilde{\pi}} + \gamma P^{\tilde{\pi}} V^{\pi} - V^{\pi} \\
\Leftrightarrow\ & V^{\tilde{\pi}} - V^{\pi} = (I - \gamma P^{\tilde{\pi}})^{-1} (R^{\tilde{\pi}} + \gamma P^{\tilde{\pi}} V^{\pi} - V^{\pi})
\end{aligned} \tag{A.8}$$

where the first equivalence results from Equation A.7, and the second from the fact that $I - \gamma P^{\tilde{\pi}}$ has an inverse. We know that $(I - \gamma P^{\tilde{\pi}})^{-1}$ and $R^{\tilde{\pi}} + \gamma P^{\tilde{\pi}} V^{\pi} - V^{\pi}$ are both nonnegative matrices. By Equation A.8, $V^{\tilde{\pi}} - V^{\pi}$ is also nonnegative, which concludes the proof. □