Task-Driven Discretization of the Joint Space of Visual Percepts and Continuous Actions

Sébastien Jodogne and Justus H. Piater

University of Liège — Montefiore Institute (B28)
B-4000 Liège, Belgium
{S.Jodogne, Justus.Piater}@ULg.ac.be

Abstract. We target the problem of closed-loop learning of control policies that map *visual* percepts to *continuous* actions. Our algorithm, called *Reinforcement Learning of Joint Classes* (RLJC), adaptively discretizes the joint space of visual percepts and continuous actions. In a sequence of attempts to remove perceptual aliasing, it incrementally builds a decision tree that applies tests either in the input perceptual space or in the output action space. The leaves of such a decision tree induce a piecewise constant, optimal state-action value function, which is computed through a reinforcement learning algorithm that uses the tree as a function approximator. The optimal policy is then derived by selecting the action that, given a percept, leads to the leaf that maximizes the value function. Our approach is quite general and applies also to learning mappings from *continuous* percepts to continuous actions. A simulated visual navigation problem illustrates the applicability of RLJC.

1 Introduction

Reinforcement Learning (RL) [1,2] is an attractive framework for the automatic design of robotic controllers. RL algorithms are indeed able to learn direct mappings from percepts to actions given a set of interactions of the robotic agent with its environment. These algorithms build on a careful analysis of a so-called *reinforcement signal* that implicitly defines the task to be solved. Using RL potentially simplifies the design process, as real-world robotic applications are in general difficult to model and to solve directly in a programming language.

Unfortunately, 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 learn direct mappings from percepts to *continuous* actions. This is in contrast to continuous perceptual spaces, for which many solutions exist. The challenge of continuous actions spaces arises from the fact that standard update rules based upon Bellman's optimality equations are only applicable on finite sets of actions, as they 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.

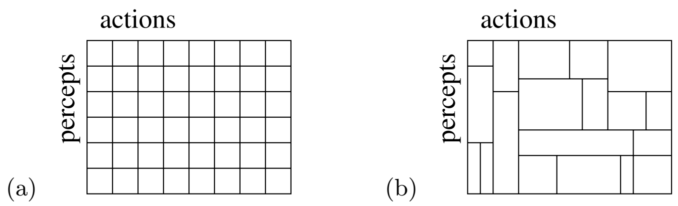

Fig. 1. Illustration of the discretization process of (a) RLVC, and (b) RLJC

Previously-investigated solutions for handling continuous actions without *a priori* discretization generally use function approximators such as neural networks [3], tile coding [4], or wire fitting [5]. However, to the best of our knowledge, none of these methods can cope simultaneously with high-dimensional, *discrete* perceptual spaces. As a consequence, vision-based robotic tasks with continuous output such as visual servoing cannot currently be solved through RL. This paper presents the *Reinforcement Learning of Joint Classes* (RLJC) algorithm, which enables such closed-loop learning of direct mappings from images to continuous actions. RLJC is a generalization of the *Reinforcement Learning of Visual Classes* (RLVC) algorithm [6] to continuous actions.

RLJC discretizes the problem space by applying tests in the input perceptual space *and* in the output action space, i.e. by testing the presence of *perceptual features* and of *action features*. For example, when the input of the agent is a binary number, suitable perceptual features could be tests on a single bit of the input. Similarly, if uni-dimensional continuous actions are considered, an action feature could be a real number that would serve as a threshold. RLJC progressively subdivides the combined percept-action (or *joint*) space, 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 constant. This way, the 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 problem space that have dissimilar properties with respect to the optimal value function. Therefore, the discretization of the action space can be inhomogeneous with respect to the perceptual space, and the action space can possibly be discretized differently at each percept. This difference is illustrated in Figure 1.

The idea of discretizing the joint space is also present in the *JoSTLe* algorithm [7], an extension of *Variable Resolution Grids* [8]. However, JoSTLe is specifically designed for *continuous* perceptual spaces, as it heavily relies on Kuhn triangulations of the joint space. Conversely, RLJC is not limited to discrete perceptual spaces, and it can also be applied to continuous perceptual spaces. Indeed, RLJC only requires that features can be defined on the perceptual and action spaces. Therefore, one key advantage of RLJC lies in its generality. Experimental results on a simulated navigation task indicate that RLJC is a promising framework for the interactive learning of visual tasks.

2 Reinforcement Learning of Visual Classes

2.1 Theoretical Background

The *Reinforcement Learning of Visual Classes* (RLVC) [6] algorithm is first described and will serve as a basis for the *Reinforcement Learning of Joint Classes* (RLJC) algorithm[1]. RLVC is a *Reinforcement Learning* (RL) algorithm [1,2].

In RL, the environment is modeled as a set S of *states* or *percepts*[2], and the agent interacts with it through a set A of *actions*. The environment obeys a stationary discrete-time dynamics: If at time t, the agent takes the action a_t while the environment lies in a state s_t, the state s_{t+1} is reached with probability $\mathcal{T}(s_t, a_t, s_{t+1})$. A stationary *reinforcement signal* $\mathcal{R} : S \times A \mapsto \mathbb{R}$ gives a quantitative evaluation of taking an action in the presence of a percept. This signal is possibly delayed, meaning that a good (resp. bad) reaction is not required to be rewarded (resp. penalized) immediately. Therefore, an *interaction* with the environment is summarized as a quadruple $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$. If S and A are finite, the quadruple $\langle S, A, \mathcal{T}, \mathcal{R}\rangle$ is known as a *Markov Decision Process* (MDP).

A stationary *percept-to-action mapping* (or *control policy*) is a function $\pi : S \mapsto A$ that links the percepts to the actions. Any control policy π induces a *value function* $V^\pi : S \mapsto \mathbb{R}$ that corresponds to the expected discounted return over time if that policy is followed from a given percept $s \in S$:

$$V^\pi(s) = \mathrm{E}^\pi\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s\right\}, \text{ for each } s \in S, \tag{1}$$

where $\gamma \in [0, 1[$ is the *discount factor* that gives the current value of the future reinforcements. The goal of RL is to learn an *optimal policy* π^* that maximizes the value function for all the percepts $s \in S$. The value function V^* of an optimal policy π^* is unique and is called the *optimal value function*. RL algorithms are able to extract an optimal policy π^* from a database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$ without relying on any knowledge of $\mathcal{T}$ or $\mathcal{R}$.

Another useful concept is that of the *state-action value function* $Q^\pi : S \times A \mapsto \mathbb{R}$ of a policy π. Such a function provides a convenient way to embed, in a single framework, the dynamics of the environment and the value function V^π. For each state $s \in S$ and each action $a \in A$, $Q^\pi(s, a)$ is the expected discounted return obtained by starting from state s, taking action a, and thereafter following π:

$$Q^\pi(s, a) = \mathrm{E}^\pi\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a\right\}, \text{ for each } s \in S \text{ and } a \in A. \tag{2}$$

The (unique) *optimal state-action value function* Q^* is defined as the state-action value function of an optimal policy π^*. Once Q^* is known, it is possible to extract the optimal value function V^*, as well as an optimal policy π^* by choosing for each $s \in S$: $V^*(s) = \sup_{a\in A} Q^*(s, a)$ and $\pi^*(s) = \operatorname{argsup}_{a\in A} Q^*(s, a)$.

[1] The formalism is different from that originally used to describe RLVC [6]. This allows us to unify RLVC and RLJC within a single theoretical framework.

[2] More explicitly, we assume that the perceptual space is fully observable.

2.2 Incremental Discretization of the Perceptual Space

Because standard RL algorithms rely on a tabular representation of the value functions, they quickly become impractical as the number of possible percepts increases. This is evidently a problem in visual tasks. In RLVC, we have proposed to constrain the allowed structure of the state-action value functions $Q(s, a)$ by resorting to a *percept classifier* $\mathcal{C}$ that discretizes the perceptual space S into a finite set of *perceptual classes* $\{c_1, \ldots, c_k\}$ by testing the presence of features in the percepts. RLVC assumes the finiteness of the action space: $A = \{a_1, \ldots, a_m\}$.

Formally, let F_S be a (possibly infinite) set of *perceptual features* that can be defined on the perceptual space. Perceptual features suitable for visual tasks are discussed in Section 2.5. These features are required to be binary: Given a percept and a perceptual feature, the feature is either present in the percept or not. Therefore, the existence of a *perceptual feature detector* is assumed, which is a Boolean function $\mathcal{D}_S : S \times F_S \mapsto \mathcal{B}$ testing whether a given percept exhibits a given perceptual feature. Furthermore, we assume the presence of a *perceptual feature generator* $\mathcal{G}_S$ that, given a percept, computes the set of all the perceptual features that are present in this percept:

$$\mathcal{G}_S : S \mapsto \mathcal{P}(F_S) : s \mapsto \{f \in F_S \mid \mathcal{D}_S(s, f)\}, \tag{3}$$

where $\mathcal{P}$ denotes the power set.

Now, the percept classifier $\mathcal{C}$ is a binary decision tree. Each of its internal nodes is labeled by the perceptual feature, the presence of which is to be tested in that node. The n leaves of the tree define the set of perceptual classes $\{c_1, \ldots, c_n\}$. To classify a percept, the system starts at the root node, then progresses down the tree according to the result of the perceptual feature detector $\mathcal{D}_S$ for each perceptual feature found during the descent, until it reaches a leaf.

Once a percept classifier $\mathcal{C}$ is fixed, all the percepts $s, s' \in c_i$ that lie in the same perceptual class c_i are required to share the same value for any state-action value function: $Q(s, a_j) = Q(s', a_j)$, for any action $a_j \in A$. Therefore, for a percept classifier $\mathcal{C}$ that induces n perceptual classes, any state-action value function $Q(s, a)$ is approximated as a function

$$\widetilde{Q}_{\mathcal{C}}(s, a, \boldsymbol{r}) = \boldsymbol{r}[i, j], \text{ if } \mathcal{C}(s) = c_i \text{ and } a = a_j, \tag{4}$$

where $\boldsymbol{r} \in \mathbb{R}^{n \times m}$ is a matrix of free parameters whose dimension is equal to the number of perceptual classes in $\mathcal{C}$ times the number of possible actions.

RLVC starts with a binary decision tree $\mathcal{C}_0$ that consists of a single leaf. Such a percept classifier maps all the percepts to the same perceptual class. Then, RLVC computes a matrix of parameters $\boldsymbol{r}_0^*$ that defines the optimal state-action value function $Q_0^*(s, a) = \widetilde{Q}_{\mathcal{C}_0}(s, a, \boldsymbol{r}_0^*)$ that is induced by $\mathcal{C}_0$. As the perceptual space is discretized, this can be done using standard RL algorithms [6]. Of course, the optimal decisions cannot always be made using Q_0^*, as percepts requiring different reactions are associated with the same class: $\mathcal{C}_0$ introduces *perceptual aliasing* [9], and the agent must refine the aliased class. So, the agent dynamically selects a new *distinctive* perceptual feature, i.e. one that best disambiguates the aliased percepts with respect to Q^*_0. This selection process is described in Sections 2.3 and 2.4. Then, the selected perceptual feature is used to refine the percept classifier $\mathcal{C}_0$, leading to a new classifier $\mathcal{C}_1$, and the process iterates.

To summarize, RLVC builds a sequence $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots$ of growing decision trees, in a sequence of attempts to remove perceptual aliasing. An optimal state-action value function Q^*_k is computed for each percept classifier $\mathcal{C}_k$ in the sequence. At each step k, some leaves are replaced by tests on highly informative features, and the number of perceptual classes in the classifier grows.

2.3 Detecting Perceptual Aliasing

RLVC uses Bellman residuals to detect the perceptual classes that are aliased in a percept classifier $\mathcal{C}_k$. Bellman's optimality equation states that, if Q^* is the optimal state-action value function of the controlled system, then:

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \cdot \sum_{s' \in S} \mathcal{T}(s,a,s') \sup_{a' \in A} Q^*(s',a'), \tag{5}$$

for all $s \in S$ and $a \in A$. If A is finite, if the transition relation $\mathcal{T}$ is assumed deterministic, and if one interaction $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ is given, we deduce that:

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \cdot \max_{a' \in A} Q^*(s_{t+1}, a'). \tag{6}$$

Let now consider Q^*_k, the optimal state-action value function induced by $\mathcal{C}_k$. As a consequence of Equations 4 and 6, for any time stamp t in the database of interactions, if a_t corresponds to the jth action in the finite set A, the scalar

$$\begin{aligned} \Delta_t &= Q^*_k(s_t, a_t) - r_{t+1} - \gamma \cdot \max_{a' \in A} Q^*_k(s_{t+1}, a') && (7) \\ &= \boldsymbol{r}^*_k \left[\mathcal{C}_k(s_t), j\right] - r_{t+1} - \gamma \cdot \max_{a' \in \{1,\ldots,m\}} \boldsymbol{r}^*_k \left[\mathcal{C}_k(s_{t+1}), a'\right] && (8) \end{aligned}$$

is called the *Bellman residual* at time t, and is a measure of the perceptual aliasing occurring in the perceptual class $\mathcal{C}_k(s_t)$. If the environment is deterministic and if the percept classifier $\mathcal{C}_k$ is free of aliasing, then Δ_t should always be zero.

Let c_i be a perceptual class that belongs to the percept classifier $\mathcal{C}_k$, and let $a \in A$ be an action. The set $T_k(c_i, a)$ is defined as the time stamps of all the interactions that are simultaneously related to the class c_i and to the action a:

$$T_k(c_i, a) = \{t \mid \mathcal{C}_k(s_t) = c_i \text{ and } a_t = a\}. \tag{9}$$

Following the reasoning above, the perceptual class c_i is considered aliased with respect to an action $a \in A$ if the set of Bellman residuals $\{\Delta_t \mid t \in T_k(c_i, a)\}$ has a variance that exceeds a given threshold $\tau \in \mathbb{R}^+_0$.

2.4 Selecting Distinctive Perceptual Features

We now turn to the problem of selecting a distinctive feature that best disambiguates an aliased perceptual class c_i in the percept classifier $\mathcal{C}_k$ with respect to an action $a \in A$. To this end, we extract all the perceptual features that can be generated from the interactions that are simultaneously related to c_i and a:

$$F_k(c_i, a) = \{f \mid (\exists t \in T_k(c_i, a))\ f \in \mathcal{G}_S(s_t)\}. \tag{10}$$

Among this set of candidate perceptual features, we select the feature that best explains the variations in the set of Bellman residuals. This is a regression problem, for which we apply the popular splitting rule that is used in the CART algorithm for building regression trees [10].

Each feature $f \in F_k(c_i, a)$ splits the Bellman residuals into two parts: $\{\Delta_t \mid t \in T_k(c_i, a) \wedge \mathcal{D}_S(s_t, f)\}$ and $\{\Delta_t \mid t \in T_k(c_i, a) \wedge \neg\mathcal{D}_S(s_t, f)\}$. We select the feature $f \in F(c_i, a)$ leading to the greatest reduction in the variance of these two sub-distributions. For each candidate feature, a Student's t-test decides whether the two sub-distributions of Bellman residuals are significantly different. This is important, as the transition relation $\mathcal{T}$ is in general non-deterministic, which generates variations in Bellman residuals that are not a consequence of aliasing.

2.5 Application to Visual Tasks

We now introduce perceptual features that enable RLVC to solve visual tasks [6]. Evidently, the high dimensionality and the noise of images cause problems in many fields of Computer Vision. For this purpose, the popular, highly successful *local-appearance methods* have been introduced. They postulate that, to take the right decision in a visual problem, it is often sufficient to focus one's attention only on a few interesting patterns occurring in the images.

They introduce a *visual feature transform* $\mathcal{F} : S \mapsto \mathcal{P}(\mathbb{R}^v)$, where S is the set of images, which summarizes an image as a set of *visual features* that are vectors of reals. For an image $s \in S$, $\mathcal{F}(s)$ typically contains between 10 and 1000 visual features. Most visual feature transforms have in common that: (1) they identify *interest points* in the images through specialized algorithms (Harris, Harris-affine,...) [11]; and (2) they compute a *local description* (local jets, SIFT,...) of the neighborhood of these interest points [12].

RLVC uses local descriptors as perceptual features. The set F_S of perceptual features corresponds to $\mathbb{R}^v$, and the perceptual feature detector $\mathcal{D}_S$ tests whether an image $s \in S$ exhibits some local descriptor at one of its interest points:

$$\mathcal{D}_S(s, \boldsymbol{f}) = \textbf{true} \text{ if and only if } (\exists \boldsymbol{f}' \in \mathcal{F}(s))\ ||\boldsymbol{f} - \boldsymbol{f}'|| < \varepsilon, \tag{11}$$

where $\boldsymbol{f} \in F_S$ is a visual feature, and $\varepsilon \in \mathbb{R}_0^+$ is a fixed threshold. Any suitable metric $||\cdot||$ can be used to test the similarity of two visual features, e.g. the Mahalanobis distance. The corresponding perceptual feature generator $\mathcal{G}_S$ returns the local description of all the interest points in the input image: $\mathcal{G}_S(s) = \mathcal{F}(s)$.

3 Reinforcement Learning of Joint Classes

The aliasing criterion defined in Section 2.3 cannot be used anymore when the action space is continuous, for at least two reasons: The Bellman residuals (as defined by Equation 7) are unavailable, as the sup operator cannot be replaced by a max; and the set of time stamps of Equation 9 is useless, because the action space can only be sparsely sampled, so that any $T(c_i, a)$ essentially collapses to a set containing at most one element. A natural idea is therefore to also discretize the action space. RLJC follows this principle, and discretizes the *joint space* $S \times A$ instead of simply S. Whereas RLVC learns a sequence of *perceptual classifiers* $\mathcal{C}_k$ that discretize the percept space by testing perceptual features, RLJC learns a sequence of *joint classifiers* $\mathcal{J}_k$ that discretize the joint state-action space by testing features on the perceptual *and* on the action space.

Formally, a (possibly infinite) set F_A of *action features* is introduced in addition to the set F_S of perceptual features. Just as perceptual features, the action features are required to be binary, and the presence of an *action feature detector* $\mathcal{D}_A : A \times F_A \mapsto \mathcal{B}$ that tests the presence of an action feature in an action is assumed, as well as the presence of an *action feature generator* $\mathcal{G}_A : A \mapsto \mathcal{P}(F_A)$ that computes the action features that a given action exhibits.

3.1 Features for Continuous Action Spaces

We are interested in closed-loop learning of mappings from images to continuous actions. Thus, $A = \mathbb{R}^a$ for some positive number a. We now introduce action features that are suitable for such a continuous space. They simply consist in testing a threshold on a particular component of the action space. Precisely, the set of action features is defined as $F_A = \mathbb{R} \times \{1, \ldots, a\}$. The corresponding action feature detector $\mathcal{D}_A$ checks whether the considered component is below the threshold or not: $\mathcal{D}_A(\boldsymbol{a}, (t, i))$ is **true** if and only if $a_i < t$. On the other hand, the action feature generator $\mathcal{G}_A$ converts an action to a action features, one for each component of the input action $\boldsymbol{a} \in \mathbb{R}^a$: $\mathcal{G}_A(\boldsymbol{a}) = \{(a_i, i) \mid i \in \{1, \ldots, a\}\}$.

3.2 Joint Features on the Percept-Action Space

Let us call $F = F_S \cup F_A$ the set of *features*, that is the union of the perceptual and of the action features. Given a state-action pair and a feature, either the feature is present in the pair or not. As a consequence, the perceptual feature detector $\mathcal{D}_S$ along with the action feature detector $\mathcal{D}_A$ trivially induces a *joint feature detector* $\mathcal{D}$ that works on the joint space, and that is defined as:

$$\mathcal{D} : (S \times A) \times (F_S \cup F_A) \mapsto \mathcal{B} : ((s, a), f) \mapsto \begin{cases} \mathcal{D}_S(s, f) & \text{if } f \in F_S, \\ \mathcal{D}_A(a, f) & \text{otherwise.} \end{cases} \tag{12}$$

Similarly, $\mathcal{G}_S$ and $\mathcal{G}_A$ can be extended to a *joint feature generator* $\mathcal{G} : (S \times A) \mapsto \mathcal{P}(F_S \cup F_A)$, by choosing $\mathcal{G}(s, a) = \mathcal{G}_S(s) \cup \mathcal{G}_A(a)$ for each $s \in S$ and $a \in A$.

In terms of this notation, a *joint classifier* $\mathcal{J}$ is a binary decision tree whose internal nodes are labeled by a feature. A joint classifier maps the (possibly infinite) joint space $S \times A$ to a finite number of *joint classes* $\{c_1, \ldots, c_n\}$ using the joint feature detector $\mathcal{D}$. Identically to the case of RLVC, such joint classifiers are thereafter used to constrain the allowed structure of the state-action value functions $Q(s, a)$. For a joint classifier $\mathcal{J}$ that induces n perceptual classes, any state-action value function $Q(s, a)$ is now approximated as:

$$\widetilde{Q}_{\mathcal{J}}(s, a, \boldsymbol{r}) = \boldsymbol{r}[\mathcal{J}(s, a)], \tag{13}$$

where $\boldsymbol{r} \in \mathbb{R}^n$ is a vector of free parameters. Note that this relation treats percepts and actions symmetrically, contrarily to Equation 4.

Very importantly, since the state-action value function $\widetilde{Q}_{\mathcal{J}}(s, a, \boldsymbol{r})$ is constrained by a joint classifier $\mathcal{J}$, the maximization step that is required by the RL algorithms is now feasible. To compute $\sup_{a' \in A} \widetilde{Q}_{\mathcal{J}}(s, a', \boldsymbol{r})$ for a percept $s \in S$, we first evaluate the set of joint classes that are *compatible* with this percept:

$$C_{\mathcal{J}}(s) = \{c_i \mid (\exists a \in A)\ \mathcal{J}(s, a) = c_i\}. \tag{14}$$

This set $C_{\mathcal{J}}(s)$ can easily be computed by a depth-first search in the binary decision tree $\mathcal{J}$: 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. Finally, as $C_{\mathcal{J}}(s)$ is obviously finite, we obtain:

$$\sup_{a' \in A} \widetilde{Q}_{\mathcal{J}}(s, a', \boldsymbol{r}) = \sup_{a' \in A} \boldsymbol{r}[\mathcal{J}(s, a')] = \max_{c_i \in C_{\mathcal{J}}(s)} \boldsymbol{r}[i]. \tag{15}$$

A similar reasoning allows the derivation of a policy from a function $\widetilde{Q}_{\mathcal{J}}(s, a, \boldsymbol{r})$.

The general scheme of RLJC is then identical to that of RLVC. A sequence of joint classifiers $\mathcal{J}_0, \mathcal{J}_1, \mathcal{J}_2, \ldots$ is generated, starting with a joint classifier $\mathcal{J}_0$ that contains one single leaf. For each $\mathcal{J}_k$ in the sequence, the optimal state-action value function Q^*_k constrained by $\mathcal{J}_k$ is computed, thanks to an algorithm that is described in Section 3.3. Then, some informative features are selected by relying on an analysis of the Bellman residuals that are induced by Q^*_k. The corresponding process is described in Section 3.4. 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.

3.3 Optimal State-Action Value Functions in the Joint Space

At each step k, the function Q^*_k is to be computed given the database of interactions $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ that is the input of RLJC. As the structure of Q^*_k is constrained by Equation 13, this amounts to computing a vector $\boldsymbol{r}^*_k \in \mathbb{R}^{n_k}$, where n_k is the number of joint classes in $\mathcal{J}_k$ [1].

For this purpose, we use the *Fitted Q Iteration* algorithm [13], that generalizes the *Value Iteration* algorithm [2]. It uses an arbitrary family of nonparametric function approximators. Therefore, the existence of an oracle called `learn` is assumed. Given a database of samples $\langle s_t, a_t, v_t \rangle$, where s_t is a state, a_t is an action and v_t is a real number, `learn` builds a function approximator that represents a state-action value function $Q : S \times A \mapsto \mathbb{R}$ that is the closest possible to the given sample distribution. The algorithm computes a sequence $Q_0, Q_1, \ldots, Q_i$ of state-action value functions, starting with $Q_0 = \texttt{learn}(\{\langle s_t,\ a_t,\ r_{t+1} \rangle\})$. Equation 6 is then turned into an update rule that makes calls to the oracle:

$$Q_{i+1} = \texttt{learn}\left(\left\{\langle s_t,\ a_t,\ r_{t+1} + \gamma \cdot \sup_{a' \in A} Q_i(s_{t+1}, a')\rangle\right\}\right). \tag{16}$$

The algorithm stops when $Q_i \approx Q_{i+1}$. By virtue of Bellman's equations, it is possible to show that $Q_i \approx Q^*$ after convergence [13].

In our framework, RLJC directly uses the nonparametric function approximators $\widetilde{Q}_{J_k}(s, a, \boldsymbol{r})$ that are defined by Equation 13. Given a set of samples $\langle s_t, a_t, v_t\rangle$, the corresponding `learn` oracle computes a vector $\boldsymbol{r}$ that simply averages the values of the samples over the joint classes that are defined by $\mathcal{J}_k$: $\boldsymbol{r}[j] = \mu(\{v_t \mid \mathcal{J}_k(s_t, a_t) = c_j\})$, for each $j \in \{1, \ldots, n_k\}$, where $\mu(\cdot)$ denotes the mean of a set of reals. The maximization over the action space that is present in the update rule is achieved through Equation 15. When Fitted Q Iteration has completed the generation of the sequence $\boldsymbol{r}^{(0)}, \boldsymbol{r}^{(1)}, \ldots, \boldsymbol{r}^{(i)}$, the parameter $\boldsymbol{r}_k^*$ is set to $\boldsymbol{r}^{(i)}$, which defines the optimal state-action value function Q_k^*.

3.4 Detecting and Removing Aliasing in the Joint Space

The algorithms that were presented for selecting new features are now adapted to continuous action spaces (cf. Sections 2.3 and 2.4). Thanks to Equation 15, the definition of Bellman residuals of Equation 7 can be further expanded:

$$\Delta_t = \boldsymbol{r}[\mathcal{J}(s_t, a_t)] - r_{t+1} - \gamma \cdot \max_{c_i \in C_{\mathcal{J}}(s_{t+1})} \boldsymbol{r}[i]. \tag{17}$$

Once again, if the environment is deterministic and if the percept classifier $\mathcal{J}_k$ is free of aliasing, these residuals should be zero. Let c_i be a joint class of $\mathcal{J}_k$. Just as in RLVC, we define the set $T_k(c_i)$ of time stamps of interactions that are related to the class c_i, and the set $F_k(c_i)$ of candidate features for this class:

$$T_k(c_i) = \{t \mid \mathcal{J}_k(s_t, a_t) = c_i\}, \tag{18}$$

$$F_k(c_i) = \{f \mid (\exists t \in T_k(c_i))\ f \in \mathcal{G}(s_t, a_t)\}. \tag{19}$$

In terms of these definitions, the aliasing criterion and the feature selection process can be adapted to the joint space. A joint class c_i of the joint classifier $\mathcal{J}_k$ is considered aliased if the set of residuals $\{\Delta_t \mid t \in T_k(c_i)\}$ has a variance that exceeds a threshold $\tau \in \mathbb{R}_0^+$. If c_i is considered aliased, RLJC then selects the candidate feature inside $F_k(c_i)$ that most reduces the variance in the two sub-distributions of Bellman residuals that are induced by the feature. A Student's t-test is also applied, to make RLJC robust to non-deterministic environments.

4 Experimental Results

RLJC has been evaluated on an abstract task that parallels a real-world scenario while avoiding any unnecessary complexity. This task is depicted in Figure 2 (a). 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. Its goal is

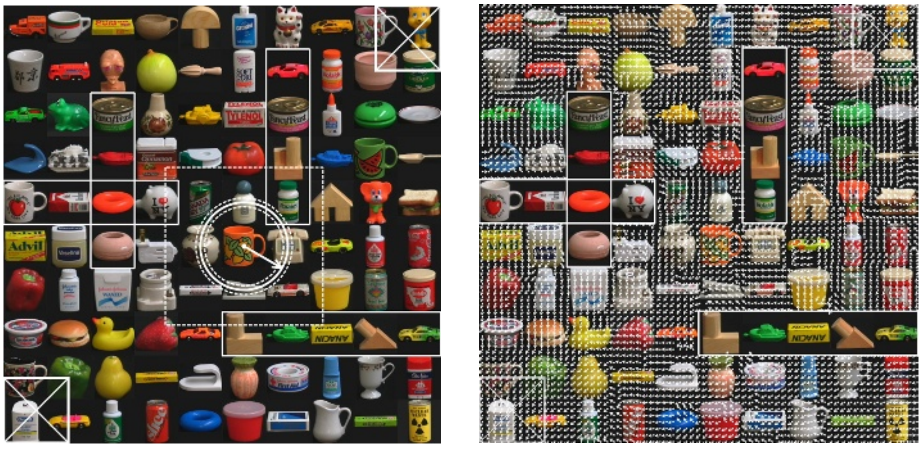
(a) (b)

Fig. 2. (a) A 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. 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. (b) The resulting image-to-action mapping $\pi^* = \text{argsup}_{a \in A} Q^*_k(s, a)$, sampled at regularly-spaced points. RLJC manages to choose the correct action at most locations[4].

to reach as fast as possible one of the two exits of the maze. 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, the standard deviation of which 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.

The agent earns a reward of 100 when an exit is reached. Any other move, including the forbidden ones, generates zero reinforcement. In this task, γ was set to 0.9. When the agent succeeds at escaping the maze, it reaches a terminal state. Note that the agent is faced with 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 a montage of pictures from the COIL-100 database[5]. 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, as long as too small displacements are not considered. Importantly, the walls are transparent, so that the sensors also return the portions of the tapestry that are behind them. Therefore, the agent cannot directly locate the walls.

[4] A full-sized version of this image is available for download at: `http://www.montefiore.ulg.ac.be/~jodogne/papers/rljc-policy.pdf`

[5] `http://www.cs.columbia.edu/CAVE/coil-100.html`

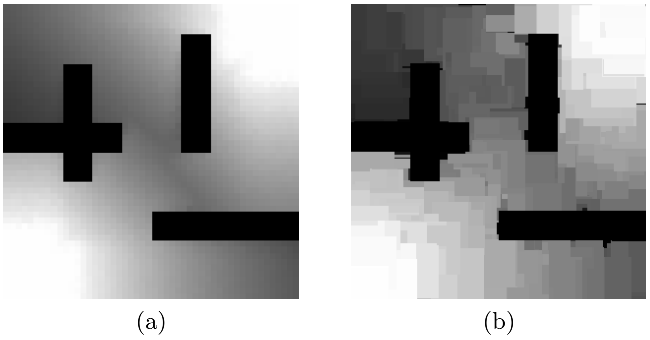

Fig. 3. (a) The optimal value function, if the agent has direct access to its (x, y) position, if the set of possible locations is discretized into a 50×50 grid, and if the set of actions is discrete and contains 4 actions (go up, down, left or right). The brighter the location, the greater its value. (b) The final value function obtained by RLJC.

In this experiment, SIFT visual features were used [14]. 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 optimal policy that results from this classifier is shown in Figure 2 (b). Figure 3 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 enough, when applied to a similar task with only four *discrete* actions, RLVC generates a perceptual classifier $\mathcal{C}_k$ that contains 205 perceptual classes [6]. In that case, $\mathcal{C}_k$ induces an optimal state-action value function that is characterized by a vector $\boldsymbol{r}_k^*$ of dimension $205 \times 4 = 820$ (cf. Equation 4). This latter number is very close to the number of joint classes that is produced by RLJC (i.e. 896). Therefore, discretizing the joint space produces a number of joint classes that corresponds to the underlying physical structure of the task.

5 Conclusions

This paper introduces *Reinforcement Learning of Joint Classes* (RLJC). RLJC is designed for closed-loop learning of mappings that directly connect visual stimuli to continuous actions that are optimal for the surrounding environment. 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 [7]. However, RLJC is more general, in the sense that it can be applied to any perceptual space and to any action space upon which it is possible to define binary 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.

Future research includes the demonstration of the applicability of our algorithms in a reactive robotic application, such as grasping objects by combining visual and haptic feedback [15]. Our current work considers tasks with complete perception and stationary environments. Of course, applying our algorithms directly on a real-world environment would raise practical problems, including partial observability, which would require the combination of our techniques with POMDP-based approaches. Another interesting open question is to test how well RLJC scales with respect to the dimensionality of the output action vector.

References

1. Bertsekas, D., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scient. (1996)
2. Sutton, R., Barto, A.: Reinforcement Learning, an Introduction. MIT Press (1998)
3. Gross, H.M., Stephan, V., Krabbes, M.: A neural field approach to topological reinforcement learning in continuous action spaces. In: Proc. of the IEEE World Congress on Computational Intelligence. Volume 3. (1998) 1992–1997
4. Santamaria, J., Sutton, R., Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces. Adaptive Behavior **6**(2) (1998) 163–218
5. Gaskett, C., Wettergreen, D., Zelinsky, A.: Q-learning in continuous state and action spaces. In: Australian Joint Conf. on Artificial Intelligence. (1999) 417–428
6. Jodogne, S., Piater, J.: Interactive learning of mappings from visual percepts to actions. In De Raedt, L., Wrobel, S., eds.: Proc. of the 22nd Intern. Conf. on Machine Learning (ICML), Bonn (Germany), ACM (2005) 393–400
7. Monson, C., Wingate, D., Seppi, K., Peterson, T.: Variable resolution discretization in the joint space. In: Intern. Conf. on Machine Learning and Applications. (2004)
8. Munos, R., Moore, A.: Variable resolution discretization in optimal control. Machine Learning **49** (2002) 291–323
9. Whitehead, S., Ballard, D.: Learning to perceive and act by trial and error. Machine Learning **7** (1991) 45–83
10. Breiman, L., Friedman, J., Stone, C.: Classification and Regression Trees. Wadsworth Intern. Group (1984)
11. Schmid, C., Mohr, R., Bauckhage, C.: Evaluation of interest point detectors. Intern. Journal of Computer Vision **37**(2) (2000) 151–172
12. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition. Volume 2., Madison (WI, USA) (2003) 257–263
13. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research **6** (2005) 503–556
14. Lowe, D.: Distinctive image features from scale-invariant keypoints. Intern. Journal of Computer Vision **60**(2) (2004) 91–110
15. Coelho, J., Piater, J., Grupen, R.: 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) (2001) 195–218