**Proceedings of the 7th IASTED International Conference
Signal Processing, Pattern Recognition and Applications (SPPRA 2010)
February 17 - 19, 2010 Innsbruck, Austria**

LEARNING SCAN PATHS FOR OBJECT RECOGNITION WITH RELATIONAL REINFORCEMENT LEARNING

Klaus Häming,
Computer Science, Visual Computing
University of Applied Sciences and Arts
Emil-Figge-Strae 42
D-44227 Dortmund, Germany
email: haeming@fh-dortmund.de

Gabriele Peters
Computer Science, Visual Computing
University of Applied Sciences and Arts
Emil-Figge-Strae 42
D-44227 Dortmund, Germany
email: gabriele.peters@fh-dortmund.de

ABSTRACT
Combining first order logic and reinforcement learning is known as relational reinforcement learning. In this work we examine the applicability of relational reinforcement learning in a computer vision task. We propose a state representation for reinforcement learning which is based on the perceived appearance of an object only, which especially makes the explicit encoding of world coordinates unnecessary. This enables computer vision researchers to endow their object recognition systems with additional flexibility and applicability, because they become independent of any knowledge about the camera parameters. In addition, we present results of an implementation of this approach. Our implementation is supported by a simple but effective image retrieval system. The image retrieval system is used to generate the reward during the learning episodes and it is described in this work as well.

KEY WORDS
Computer vision, image processing, image retrieval, relational reinforcement learning

1 Introduction

Given a camera and a set of nearly, but not completely, identical objects with a unique name attached to each. What is the best (i.e., shortest in length or steps) sequence of camera positions to discriminate these objects? That depends, of course, on the information gained at each position. If this information is based on the taken images alone, then what is the best camera path? This is the toy problem we will examine in this paper to test the applicability of relational reinforcement learning to continuous domains and computer vision.

A key issue of reinforcement learning [1] is the representation of states and actions. In this work we will propose representations of states and actions that are based on the idea of relational reinforcement learning [2]. That means that states and actions are represented as propositional clauses. Instead of encoding the position of the camera in these clauses, it is investigated how the information gained from the images alone can be sufficient. That means, it is not necessary to impose a rigid grid around the object that defines possible camera positions as seen in former works [3]. Just as we avoid an artificial discretisation of camera positions, the encoding of the actions also avoids a discretisation of the possible directions. Instead, all directions are represented as angles that are interpreted as directions roughly orthogonal to the current viewing direction.

Because these representations are continuous, the need of a regression technique for assigning Q-values is obvious. Besides, the relational representation of states and actions demand such a technique anyway. In this work, we adopt on the RIB-algorithm [4], a k-nearest-neighbor technique. Therefore, a distance measure between state-action-pairs is needed. Such a distance measure is also described in this work.

Following this section, we first review both traditional and relational reinforcement learning, then discuss the kind of reward given in our application, present our approach to modelling the states and actions, and conclude by presenting our application and its results in more detail.

2 Reinforcement Learning

Reinforcement learning (RL) [1] is a computational technique that allows an autonomous entity named agent, to learn a certain behavior using a trial and error technique. In each trial, the agent performs a sequence of actions. Each action a belongs to the set of all possible actions A. After an action is performed, the environment switches to a particular state s out of the set of all states S. If a terminal state is reached, the sequence will end. These terminal states are absorbing, meaning that all subsequent actions do not change the state. Such sequences of actions are commonly called episodes.

The agent gets a feedback from the environment after each state change. Such a feedback is a numerical value representing how desireable the current state of the environment is. It is usually called reward r and registered by the agent.

In any case, the reward function is unknown to the agent. The rewards are the only information that is taken into account when deciding on the next action to take. The simplest way to gather this information is to attach a value

$V(s)$ to each state and update it according to the received reward. The action that leads to the best state gets the highest probability to be examined further in the next episodes. This way, the agent behaves more and more optimally in the sense that it seeks to maximize the received reward while additionally spending an amount of time exploring unknown actions.

More formally, a reinforcement problem is modelled by the following components:

- A set of states S
- A set of actions A
- A transition function $\delta : S \times A \rightarrow S$
- A reward function $r : S \times A \rightarrow \mathbb{R}$

Obviously, the action to be taken in a state has to be chosen. This entity is usually called policy π. Because these policies use some means to decide on the actions, the value of the states depend on those means and therefore on the policy. Given the fixed policy π, the goal consists in the maximization of

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+1},$$

which is the sum of all rewards of the future steps (discounted by a factor γ, to stay finite). Within all possible policies, the best policy π^* generates the highest values.

$$\begin{aligned} \pi^* &= argmax_{\pi}(V^{\pi}(s_t)) \\ V^*(s_t) &= V^{\pi^*}(s_t) \end{aligned}$$

Fortunately, there is a way to compute the optimal policy without iterating through all possible policies. This is achieved by defining a function that rates state-action-pairs in contrast to the rating of states in the V^{π}-functions. This new function is called Q(uality)-function. It's definition is

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

This function can be computed iteratively by using the update equation

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a') \tag{1}$$

The last equation does not refer to a policy. Therefore, Q-learning is called a policy-free learning scheme.

3 Relational Reinforcement Learning

A serious problem in Q-learning is the so called curse of dimensionality and the poor generalization when using a simple table to store the Q-values.

The first refers to the exponential increase in the number of states with the number of state attributes and consequently the same increase in table size.

The last reflects the fact that a slight change in either state or action space renders the whole Q-table useless. Than, re-learning from scatch is required, although the problem probably has changed only slightly.

Relational reinforcement learning encodes the states using propositional logic to overcome these limitations. The exponential increase of state-action-pairs is handled by sparsely representing the Q-function through learned examples, and then applying a regression technique. There are different approaches to this, such as a decision tree [2], a kernel based appraoch [5], or a k-nearest-neighbor approach [4].

Additionally, that allows the user to teach knowledge in advance to speed up the process. This can also be used to train the RL system with a small problem first until an acceptable solution is found and then use the result as a basis for a more complex problem.

In this paper we adopt the k-nearest-neighbor approach of [4], which is now briefly reviewed.

The basic idea is to approximate the Q-value q_i of state-action-pair i by comparing the logical description of that pair to the descriptions of the examples found so far. The closest of these examples are used to derive the desired approximation $\hat{q}_i$. A suitable formula is given by

$$\hat{q}_i = \frac{\sum\limits_j \frac{1}{dist_{i,j}} q_j}{\sum\limits_j \frac{1}{dist_{i,j}}} \tag{2}$$

which is simply the weighted arithmetical mean of the nearest Q-values. The weight is given by a reciprocal distance measure $dist_{i,j}$, which depends on the logical modelling of the state-action-pairs and is defined in the following section.

4 The Markovian Property and Zero-Rewards

In general, reinforcement learning wants the states to fulfill the Markovian Property. That means, all the information necessary to predict the potential future reward has to be enclosed in the states' representation.

This requirement poses difficulties for applications aiming at learning object properties from moving a camera around them, because the future camera movement has to depend on the information gathered so far.

If we base our reward at each step on the information that is gathered from the observed object, then the degree of novelty of this information should substantially contribute to the height of the reward. This kind of reasoning is tempting, but must be considered a trap. Because the RL approach seeks to maximize the reward of a sequence of actions, such a procedure will result in the agent scanning the object completely until no new information can be found, and then heading to the goal state.

Therefore, we do not reward a simple movement at all. The only action that receives a reward is the goal state.

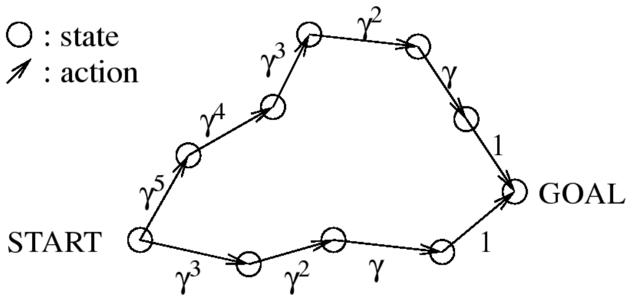

Figure 1. A discount factor in an RL setting with 0 reward everywhere except for the goal state. This leads to a preference of short paths.

If we look at equation (1), we will now notice the importance of $\gamma \in]0;1[$. If we set $\gamma := 1$, all Q-values will approach 1, since r is always 0. But with $\gamma \in]0;1[$ we will find a greater Q-value attached to states closer to the goal than to those farther away. This is roughly depicted in figure 1 and leads to the desired preference of short paths towards the goal state.

It may be thrown in that negative rewards pose a third option. And similar to the zero-reward approach the shortest path to the goal state is expected to be taken. This time however, the distance is modified by the negative rewards and may be used to indicate a path somehow easier to follow. This is left for future investigations.

5 Modelling

In an application where the entities which must be captured inside the states can assume values from a continuous domain, the design of the state representation can be difficult. As stated above, it is important to keep the state space small, because the number of possible paths from one state to the final state increases exponentially. This is also known as "the curse of dimensionality". The usual solution in an application with moving cameras is to limit the camera to certain positions on a sphere [3].

In contrast, our approach will allow arbitrary positions, because those will be encoded only implicitly by the appearance of the object in terms of visible features. The information captured is depicted in figure 2, and an encoding of the view looks like this:

$$\texttt{inC(FP1)} \wedge \texttt{inC(FP6)} \wedge \texttt{inC(FP5)} \wedge \texttt{inB(FP5)} \wedge \texttt{inA(FP3)} \wedge \texttt{inA(FP17)}$$

The action part of the state-action-pair is captured as a direction. The clause describing this is

$$\texttt{to}(\alpha)$$

where the parameter is simply an angle $\alpha \in [0; 2\pi]$.

To aspects have to be mentioned in this context. Firstly, it is unclear from which directions to choose and

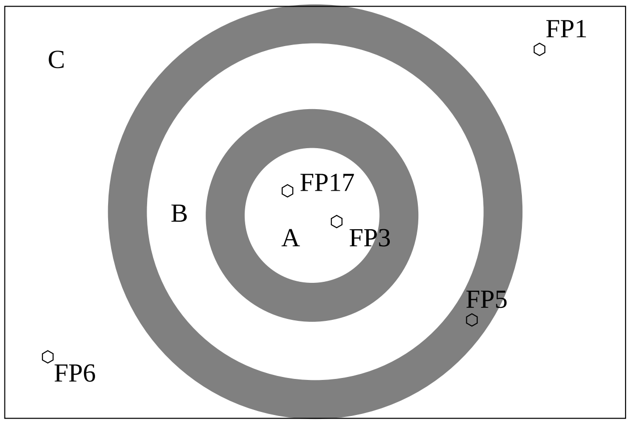

Figure 2. Relevant information captured in a state description.

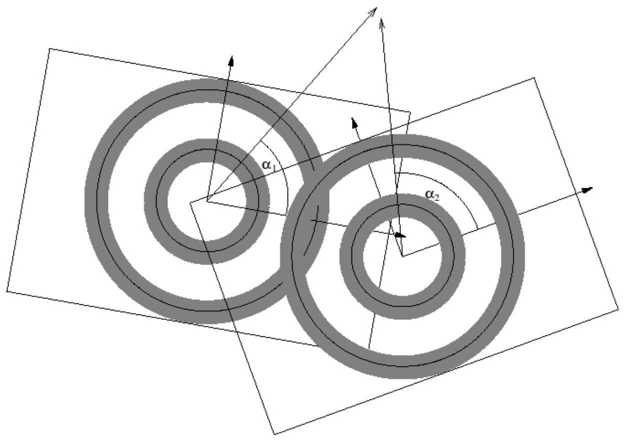

Figure 3. The similarity of actions is measured by the distance of the camera centers after moving them in the direction of the angle in their corresponding `to`-clause.

whether the distribution of the visible features should have an impact on this choice. Whether or not this would be beneficial will be tested by adding clauses that describe the feature density in certain parts of the image in some of the experiments.

$$\texttt{densityH1}(\beta_1) \wedge \texttt{densityH2}(\beta_2) \wedge \texttt{densityL1}(\beta_3) \wedge \texttt{densityL2}(\beta_4)$$

These clauses encode the directions in which the two highest and lowest feature densities have been found, respectively.

Secondly, we have to cope with the disadvantage that the direction is 2D only, missing a component describing the distance to the object. To overcome this limitation, the observer in our experiments is capable of stereovision, trying to maintain a fixed distance.

As stated above, equation (2) includes a reference to a function called $dist_{i,j}$ which measures the distance between two state-action-pairs. To make things easier, our distance function is based on a similarity measure $sim_{i,j} : (i, j) \to [0; 1]$.

$$dist_{i,j} = 1 - sim_{i,j}$$

This similarity measure is defined as a product of the state similarity and the action similarity.

$$sim_{i,j} = sim^S_{i,j} \cdot sim^A_{i,j}$$

with

$$\begin{aligned} sim^S_{i,j} &= f\left(\sum_{X \in \{\{a,b\} | a,b \in C\}} w_{\{a,b\}} s_{i,j,\{a,b\}} \right) \\ sim^A_{i,j} &= f(s_{i,j,\texttt{to}}) \end{aligned}$$

and $C := \{inA, inB, inC\}$.

The $s_{i,j,X}$ functions measure the similarities. The weights $w_{\{a,b\}}$ control the influence of the different aspects of the state representation on the distance measure.

The function f is used to adjust the punishment linked to $s_{i,j,X}$ and to ensure the range is limited to $[0; 1]$. We use a sigmoid function for this purpose and to additionally impose a soft threshold to punish bad values quickly while allowing small deviations from the optimum:

$$f(x) = \frac{1}{1 + \mathrm{e}^{-s(x-i)}}$$

where s is the steepness of the slope (set to 100 in our tests) and i is the inflection point (set to 0.9).

This leaves the similarities $s_{i,j,X}$ to define. The simpler of those is the similarity $s_{i,j,\texttt{to}}$, the distance according to the different `to`-clauses. Two of these clauses point to the same state if the camera captures the same features after moving in that direction. To skip the actual computation of this, we simply use the distance of the camera centers after moving them along the directions the actions point to. The approach is depicted in figure 3. Of course, this procedure is only valid, if the viewing direction of the cameras are similar. If this is not the case, the state similarity $sim^S_{i,j}$ would be low in the first place resulting in a low overall similarity value anyway.

The similarity component according to states is a similarity measure of image content. This is slightly more complex. The idea is to base its calculation on the amount of overlap. The more one image overlaps with another, the more similar the images will be considered. For example, if the A-regions overlap, the states can be considered similar.

The implementation of this idea uses a simple scheme for the values $w_{\{a,b\}}$ applying the following weights:

$w_{\{a,b\}}$		a		
		inA	inB	inC
b	inA	1.0	0.5	0.3
	inB	0.5	0.3	0.2
	inC	0.3	0.2	0.1

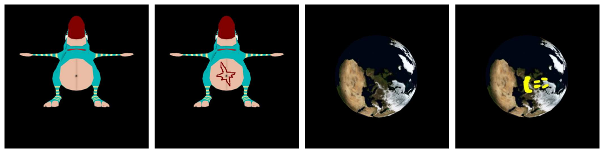
Figure 4. Example Objects.

Since the area under the center region is relatively small, these features get a significant higher weight.

Finally, the similarities $s_{i,j,\{a,b\}}$ are calculated as

$$s_{i,j,\{a,b\}} = \frac{|\texttt{cl}_a(i) \cap \texttt{cl}_b(j)|}{|\texttt{cl}_a(i) \cup \texttt{cl}_b(j)|}$$

where $\texttt{cl}_X(i)$ maps the index of state-action-pair i to the set of all its X-clauses. This function simply relates the cardinality of the intersection with the cardinality of the union.

6 Application

The previously described framework has been used to build an application that learns to discriminate two similar objects. These objects look essentially the same except for a minor difference, for example in texture as seen in figure 4. It is learned how to perform an effective scan of an object to tell which one in the database it belongs to. The database was created by scanning the objects from many viewpoints and calculating visible features and their descriptors for each viewpoint.

6.1 Feature and Descriptor Calculation

To recognize and match feature points througout images taken from different camera positions, they are required to be reasonable stable. To comply with this requirement, we use a scale space Harris detector, combining the real time scale space of Lindeberg [6] with the Harris detector [7], while modifying the latter to make it work in a scale space pyramid [8].

The descriptors attached to these feature points are SIFT descriptors [9]. While they only use those feature coordinates that pose maxima in scale space, we take all coordinates into account (as long as they are maxima in their scales). This overcomes some stability issues with the original approach [10].

As a result we get up to 500 features in each image. To reduce this amount, each camera takes a second image from a slightly different view point. Then we apply a feature matching using a kd-tree [11]. The resulting correspondences are filtered by applying the epipolar constraint [12]. Only those feature points that survive this procedure are stored in our database. We aim at about 100 features for each view. Figure 5 shows an example matching of two different views of the globe.

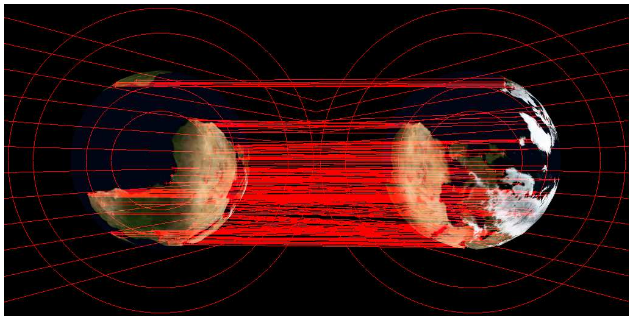

Figure 5. An example for matching is displayed. The correspondences, the epipolar geometry, and the circles representing the different feature regions are depicted. Note the wrong correspondences. Though they are the best found along the epipolar lines, the score of this matching is easily excelled by a more similar view.

6.2 Reward

As noted in section 4, the reward is zero for all actions except those reaching the goal state. In that case, the reward is one. In each state an image Img is captured. After computing features and their descriptors of this image, the descriptors are used to identify the goal state. This identification is a two phase process.

Phase 1 iterates through the objects inside the database and identifies for each object the most similar view. This is basically an image retrieval task. Given one object Obj, our approach begins with building a kd-tree KDT_{Obj} with all descriptors belonging to Obj. For each descriptor $D_{Img,i}$ of Img, the most similar descriptor $D_{Obj,j}$ in KDT_{Obj} is identified, and its Euclidean distance $d(D_{Img,i}, D_{Obj,j})$ is computed. This distance is used to vote for the view $D_{Obj,j}$ belongs to. A distance of zero scores a full vote, a distance greater than zero decreases the vote score subject to

$$\texttt{score}(d) = \frac{1}{1+d} \quad .$$

Figures 6 and 7 show resulting pairs of images. This simple approach works reasonably well, failing rarely in our experiments.

Phase 2 sorts all objects according to the score of their highest scoring view from high to low. The first object O_1 of these is returned as the result after attaching a value of certainty. This value s is computed using the following distance measure to the second best object O_2:

$$s = \frac{\texttt{score}(O_1) - \texttt{score}(O_2)}{\texttt{score}(O_1)}$$

If this value exceeds a certain threshold (e.g., 0.25), the object is considered identified, and the current episode ends.

Figure 8 shows the development of the sample set constituting the Q-function. After the first episode only

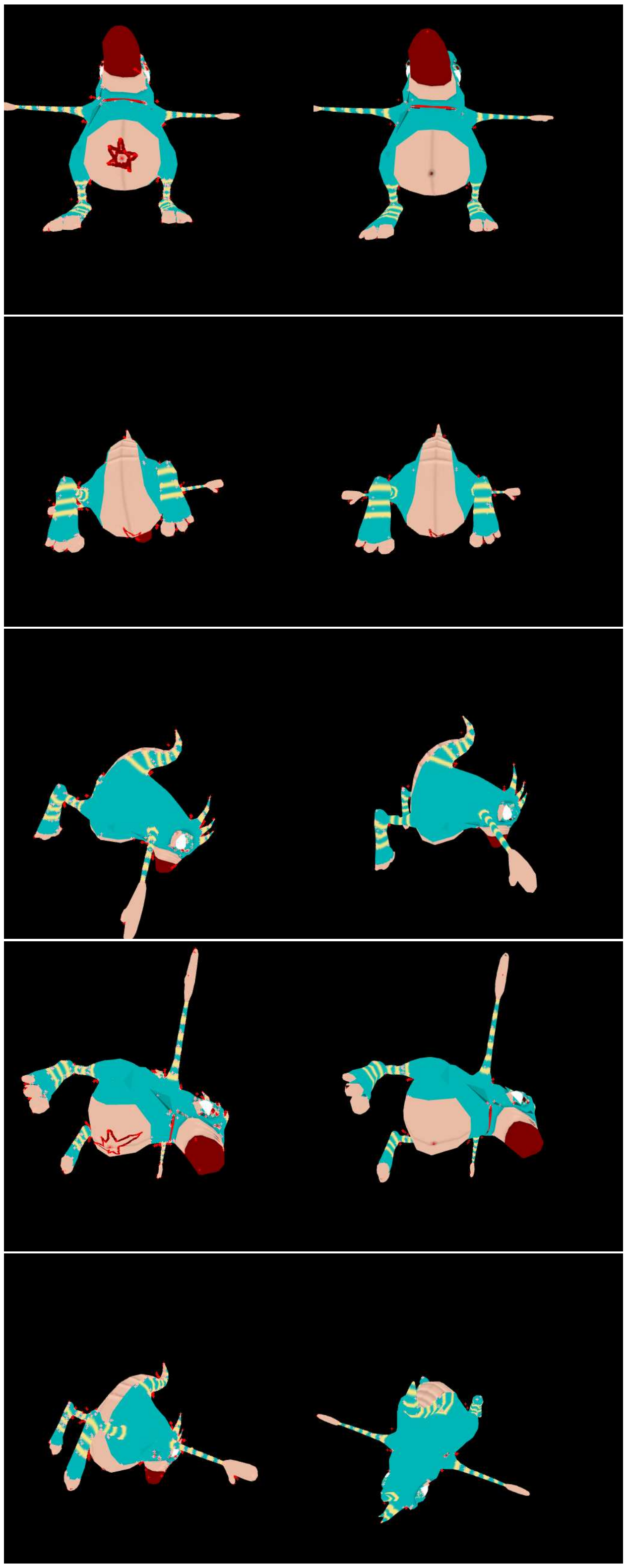

Figure 6. Image retrieval examples. The difference between the objects is the tattoo around the belly button. The left image is observed by the agent and the right is the most similar according to our retrieval algorithm. The last example is assigned wrongly, but has also a very low confidence.

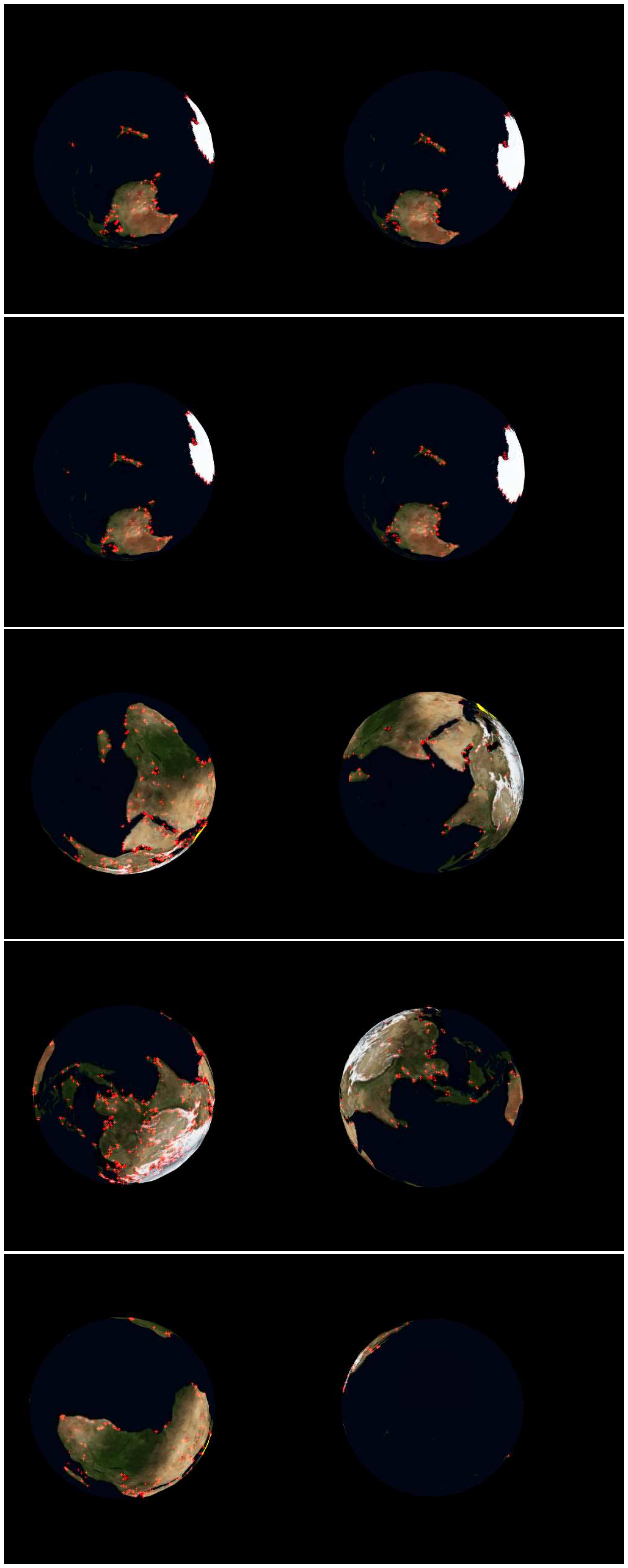

Figure 7. Image retrieval examples. The first two slightly different views are assigned to the same database view. The third example is an assignment to a rotated view. This is possible because the SIFT descriptors are rotationally invariant. The last one is, again, clearly a mistake, but has a low confidence value.

three samples are found inside the sample set. The beginning of the development of the Q-function is easily verified. The first sample approximates the Q-function as a constant 0, until the first step encounters a reward of 1. Then, two additional samples are inserted: the state-action-pair that led to the goal state and the goal state itself. Further episodes insert samples according to the same rules as in the original RIB-algorithm [4].

7 Conclusion and Future Work

We have presented first results of a vision system that is able to discriminate similar objects. It uses the technique of relational reinforcement learning. The results can be regarded as preliminary as we have carried out a small number of experiments only and also used a limited image data base. We also described a simple image retrieval system and object recognition scheme that are used to generate the rewards and mark the end of an episode.

This work leads to a number of possible future extensions. First of all the image retrieval using a simple kd-tree is quite slow and needs an acceleration. This is because we used 128 images containing about 100 feature points for each object. Using the best-bin-first-technique of [11] speeds up the process a little, but the matching reliability suffers quickly and values below 10000 comparisons are strongly discouraged.

Besides this performance issue, it is obvious that the data we hold is vastly redundant. We plan to establish a feedback loop to use the learned data to indicate views or feature sets that should be shared across objects.

Integrating the generation of the object database into the learning algorithm would result in an application letting the agent explore its environment and constantly adding features into the object database. Similar objects can share subsets of features. In addition, we will develop a criterion that groups feature sets as belonging to the same unique object type. In future research we will also examine the influence of the size of the image data base on the performance of our algorithm.

Acknowledgement

This research was funded by the German Research Association (DFG) under Grant PE 887/3-3.

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. MIT Press, Cambridge, 1998.

[2] S. Dzeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, *43*:7–52, 2001.

[3] G. Peters. A vision system for interactive object learning. In *IEEE International Conference on Computer*

Vision Systems (ICVS 2006), New York, USA, January 5-7, 2006.

[4] K. Driessens and J. Ramon. Relational instance based regression for relational reinforcement learning. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 123–130, 2003.

[5] T. Gartner, K. Driessens, and J. Ramon. Graph kernels and gaussian processes for relational reinforcement learning. In *Inductive Logic Programming, 13th International Conference, ILP*, 2003.

[6] T. Lindeberg and L. Bretzner. Real-time scale selection in hybrid multi-scale representations. In *Proceedings Scale-Space*, volume 2695 of *Springer Lecture Notes in Computer Science*, pages 148–163, 2003.

[7] C. Harris and M. Stephens. A combined corner and edge detector. In *4th ALVEY Vision Conference*, pages 147–151, 1988.

[8] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, *60*(1):63–86, 2004.

[9] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, *60*(2):91–110, 2004.

[10] A. Baumberg. Reliable feature matching across widely separated views. *CVPR*, 01:1774, 2000.

[11] J. Beis and D. Lowe. Shape indexing using approximate nearest-neighbor search in highdimensional spaces. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, pages 1000–1006, 1997.

[12] R. I. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, second edition, 2004.

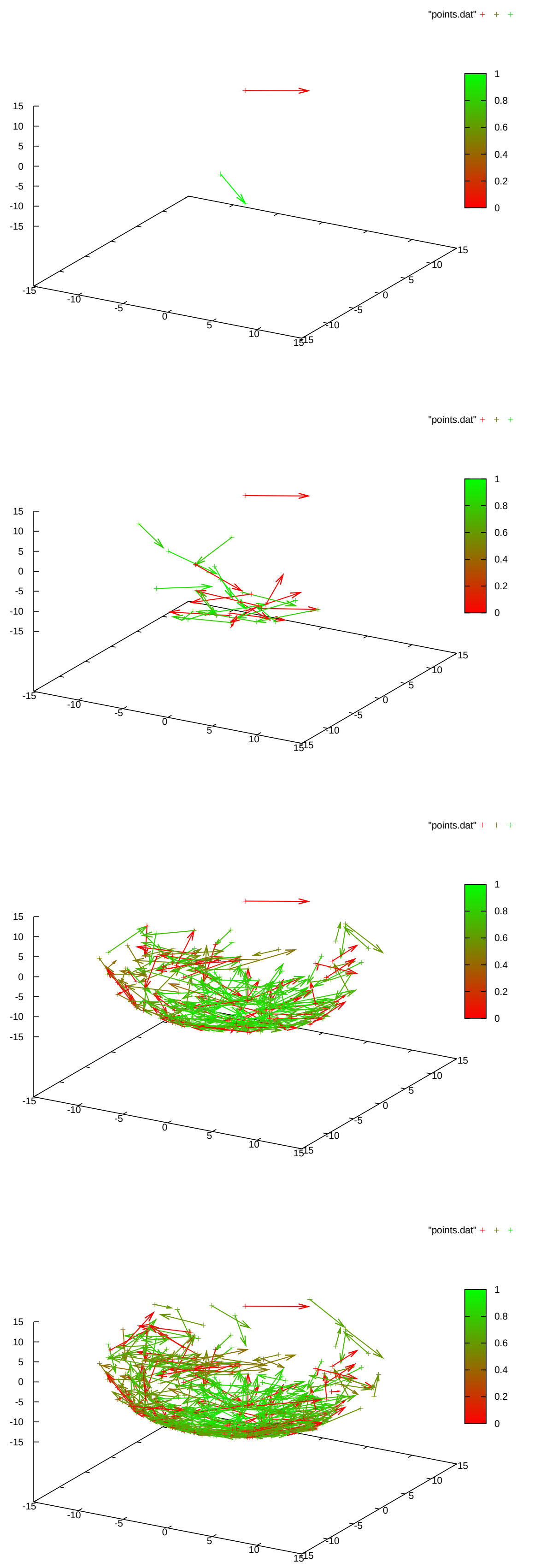

Figure 8. Samples used to approximate the Q-Function after 1, 2, 5, and 10 episodes (top to bottom).