Cost Integration in Multi-Step Viewpoint Selection for Object Recognition

Christian Derichs[1*], Frank Deinzer[2], and Heinrich Niemann[1]

[1] Chair for Pattern Recognition, Department of Computer Science,
Universität Erlangen-Nürnberg, Martensstr. 3, 91058 Erlangen
{derichs,niemann}@informatik.uni-erlangen.de,
[2] Siemens Medical Solutions, Siemensstr. 1, 91301 Forchheim,
frank.deinzer@siemens.com

Abstract. During the last years, computer vision tasks like object recognition and localization were rapidly expanded from passive solution approaches to active ones, that is to execute a viewpoint selection algorithm in order to acquire just the most significant views of an arbitrary object. Although fusion of multiple views can already be done reliably, planning is still limited to gathering the next best view, normally the one providing the highest immediate gain in information. In this paper, we show how to perform a generally more intelligent, long-run optimized sequence of actions by linking them with costs. Therefore it will be introduced how to acquire the cost of an appropriate dimensionality in a non-empirical way while still leaving the determination of the system's basic behavior to the user.
Since this planning process is accomplished by an underlying machine learning technique, we also point out the ease of adjusting these to the expanded task and show why to use a multi-step approach for doing so.
Keywords Viewpoint Selection, Active Vision, Reinforcement Learning

1 Introduction

In the last few years, a lot of approaches have been made to add an active component to the task of combined object recognition and localization, leading to algorithms teaching an agent which views of an arbitrary object to take. Therefore, most of them apply quite powerful machine learning techniques to this task, like Reinforcement Learning [7], but still restrict themselves to just performing a single-step process, that is to always select the immediate best view to be taken within the next step for increasing the classification rate. Obviously this behavior might be suboptimal concerning the effort needed for reaching a certain classification reliability, e.g. if the agent passes by a viewpoint in a first step it has to take later on anyway.

In light of this drawback, the work of this paper will show how to put a more foresighted component to the viewpoint selection problem by integrating costs to the system. While further cost-integrating approaches attempt to control the effort feasible for improving the classification itself [5] or to teach the agent by introducing costs for misclassification [4], we do explicitly not aspire at an enhancement in classification.

* This work was funded by the German Science Foundation(DFG) under grant SFB 603/TP B2. Only the authors are responsible for the content.

Moreover our goal is to gain similar results, but with possibly less effort. Therefore the components of an agent's possible action have to be weighted against each other by introducing those actions' costs, respectively cost relations, which in turn have to be adjusted to the learning technique's reward modeling. Unlike [6], where those costs are determined empirically, our attention is towards acquiring them as a by-product of the training performed anyway, namely by learning and directly applying them in parallel manner to the agent-teaching, main learning process.

Thus section 2 will give a summary of the machine learning technique, namely Reinforcement Learning, underlying the viewpoint selection task. It will be directly introduced as a multi-step approach since our presented way of cost integration makes fundamental use of this assumption. As the basics are defined then, section 3 provides the method for establishing and integrating cost-factors to the already existing learning algorithms, while section 4 will introduce methods for computing the new target values more reliably and preferably without extending the processing time of the global learning process. Results concerning a cost-sensitive solution of the viewpoint selection problem will finally be shown for a co-operative environment where classification itself does not make up a critical task.

2 Viewpoint selection without costs

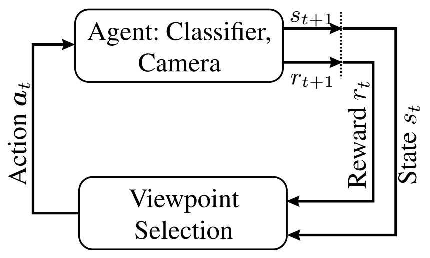

Fig. 1. Principles of Reinforcement Learning applied to viewpoint selection.

The goal of this work is to provide a solution to the problem of optimal viewpoint selection for 3D object recognition without making a priori assumptions about the objects and the classifier. The problem is to determine the next view of an object given a series of previous decisions and observations and can also be seen as the determination of a function which maps a history of observations to a new viewpoint. This function should be estimated automatically during a training step and should improve over time. Additionally, the function should take uncertainty into account in the recognition process as well as in the viewpoint selection, be classifier independent and be able to handle continuous viewpoints.

A straightforward and intuitive way to formalizing the problem is given in fig.1. A closed loop between sensing state s_t and performing action $\boldsymbol{a}_t$ can be seen. The chosen *actions* $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = (a_{t,1}, a_{t,2}, \ldots, a_{t,n})^T \tag{1}$$

correspond to the movement of the camera at time t.

The sensed states s_t, actually probability densities (2), are estimated by the employed classifier. In this paper we use a classifier [2, 3] that is able to perform a fusion of multiple acquired images. In active object recognition, a series of observed images $\boldsymbol{f}_t, \boldsymbol{f}_{t-1}, \ldots, \boldsymbol{f}_0$ of an object are given together with the camera movements $\boldsymbol{a}_{t-1}, \ldots, \boldsymbol{a}_0$ between these images. Based on these observations of images and movements one wants to draw conclusions for a non-observable state $\boldsymbol{q}_t$ of the object. This state of the object consists of its class and pose relative to the camera. In the context of a Bayesian approach, the knowledge on the object's state is given in form of the a posteriori density $p(\boldsymbol{q}_t|\boldsymbol{f}_t, \boldsymbol{a}_{t-1}, \boldsymbol{f}_{t-1}, \ldots, \boldsymbol{a}_0, \boldsymbol{f}_0)$ that is calculated by the classifier. In [2] it is discussed how to represent and evaluate this density with particle filter approaches. From the viewpoint selection's perspective, this density contains all the necessary information, so that the following *density* definition meets the Reinforcement Learning requirements:

$$s_t = p(\boldsymbol{q}_t|\boldsymbol{f}_t, \boldsymbol{a}_{t-1}, \boldsymbol{f}_{t-1}, \ldots, \boldsymbol{a}_0, \boldsymbol{f}_0) \tag{2}$$

Additionally, the classifier returns a so called *reward* r_t, which measures the quality of the chosen viewpoint. For a viewpoint that increases the information observed so far, the reward should have a large value. A well-know measure for expressing the informational content that fits our requirements is the entropy

$$r_t = -H(s_t) = -H\left(p(\boldsymbol{q}_t|\boldsymbol{f}_t, \boldsymbol{a}_{t-1}, \boldsymbol{f}_{t-1}, \ldots, \boldsymbol{a}_0, \boldsymbol{f}_0)\right) \quad . \tag{3}$$

It is important to notice that the reward should also include costs for the camera movement and object classification. This topic will be discussed in section 3.

At time t during the decision process, i.e. the selection of a sequence of viewpoints, the goal will be to maximize the accumulated and weighted future rewards, called the *return*

$$R_t = \sum_{n=0}^{\infty} \gamma^n r_{t+n+1} = -\sum_{n=0}^{\infty} \gamma^n H(s_{t+n+1}) \quad \text{with } \gamma \in [0;1] \, . \tag{4}$$

The weight γ defines how much influence a future reward at time $t+n+1$ will have on the overall return R_t. So, to meet the demands of this paper, a $\gamma > 0$ is required since we make use of a multi-step approach. Of course, the future rewards cannot be observed at time step t. Thus, the following function, called the *action-value function* $Q(s, \boldsymbol{a})$

$$Q(s, \boldsymbol{a}) = E\left\{R_t | s_t = s, \boldsymbol{a}_t = \boldsymbol{a}\right\} \tag{5}$$

is defined, which describes the expected return when starting at time step t in state s with action $\boldsymbol{a}$. In other words, the function $Q(s, \boldsymbol{a})$ models the expected quality of the chosen camera movement $\boldsymbol{a}$ for the future, if the classifier has returned s before.

Viewpoint selection can now be defined as a two step approach: First, estimate the function $Q(s, \boldsymbol{a})$ during training. Second, if at any time the classifier returns s as result, select that camera movement which maximizes the expected accumulated and weighted rewards. This function is called the *policy*

$$\pi(s) = \operatorname*{argmax}_{\boldsymbol{a}} Q(s, \boldsymbol{a}) \, . \tag{6}$$

The key issue of course is the estimation of the function $Q(s, \boldsymbol{a})$, which is the basis for the decision process in (6). One of our demands is that the selection of the most promising view should be learned without user interaction. Reinforcement Learning provides many different algorithms to estimate the action value function based on a trial and error method [8]. Trial and error means that the system itself is responsible for trying certain actions in a certain state. The result of such a trial is then used to update $Q(\cdot, \cdot)$ and to improve its policy π.

In Reinforcement Learning, a series of *episodes* are performed: Each episode k consists of a sequence of state/action pairs $(s_t^k, \boldsymbol{a}_t^k), t \in \{0, 1, \ldots, T\}$, with T *steps* at most. Each performed action $\boldsymbol{a}_t$ in state s_t results in a new state s_{t+1}. During the episode, new returns $R_t^{(k)}$ are collected for those state/action pairs $(s_t^k, \boldsymbol{a}_t^k)$ which have been visited at time t during the episode k. At the end of the episode the action-value function is updated. In our case, so called Monte Carlo Learning [8] is applied and the function $Q(\cdot, \cdot)$ is estimated by the mean of all collected returns $R_t^{(i)}$ for the state/action pair $(s, \boldsymbol{a})$ for all episodes.

As a result for the next episode one gets a new decision rule π_{k+1}, which is now computed by maximizing the updated action value function. This procedure is repeated until π_{k+1} converges to the optimal policy. The reader is referred to a detailed introduction to Reinforcement Learning [8] for a description of other ways for estimating the function $Q(\cdot, \cdot)$. Convergence proofs for several algorithms can be found in [1].

Since most of the algorithms in Reinforcement Learning treat the states and actions as discrete variables, a way to extend the algorithms to continuous Reinforcement Learning is to approximate the action-value function $\widehat{Q}(s, \boldsymbol{a})$ which can be evaluated for any continuous state/action pair $(s, \boldsymbol{a})$. Basically, this approximation is a weighted sum of the action-values $Q(s', \boldsymbol{a}')$ of all previously collected state/action pairs. All the collected action-values $Q(s', \boldsymbol{a}')$ are referred as Q-base throughout this paper. For the exact calculation of $\widehat{Q}(s, \boldsymbol{a})$ please refer to [3].

Viewpoint selection, i.e. the computation of the policy π, can now be written, according to (6), as the optimization problem

$$\pi(s) = \underset{\boldsymbol{a}}{\operatorname{argmax}} \, \widehat{Q}(s, \boldsymbol{a}) \, . \tag{7}$$

This problem (7) can be solved by global optimization techniques. In this work we use the Adaptive Random Search algorithm (ARS) [9] combined with a local simplex.

3 Integration of costs

Referring to the previous sections we first have to integrate the costs of an action into the reward of Reinforcement Learning. This is done by adding a cost term $C(\boldsymbol{a}_t)$ to the reward in (3) yielding

$$r_{t+1} = -H(s_{t+1}) - C(\boldsymbol{a}_t) \tag{8}$$

It is now necessary to take a closer look at the detailed modeling of those costs to be able to understand its influence on the system-determining reward and its interaction with $H(s_{t+1})$. In general, the description is

$$C(\boldsymbol{a}_t) = \sum\nolimits_i \kappa_i \cdot x_{t,i} = \boldsymbol{\kappa} \cdot \boldsymbol{x}_t \quad \text{with} \quad x_{t,i} = \frac{a_{t,i}}{a_{i_u}} \tag{9}$$

where index i provides a handle to the ith element of an action $\boldsymbol{a}$ an agent is actually able to perform within a given environment. Next to this, every action component $\boldsymbol{a}_{t,i}$ can be split into an amount of $x_{t,i}$ unit actions a_{i_u} for every time step. Latter ones have to be linked with a specific cost-factor κ_i in order to determine (9).

While now a_{i_u} as well as κ_i can take influence on the cost of an action we decided to let the user choose all a_{i_u} in a meaningful way beforehand, whereas κ_i will be the crucial, variable parameter for affecting the system behavior via $C(\boldsymbol{a}_t)$. Here the user is asked for i values of absolute type which might be difficult to acquire in a real world environment since they might be influenced by various conditions. But what is essential to be provided is at least a constant relation $\frac{\kappa_i}{\kappa_j}$ between all involved cost-factors, which are likely to be achieved appropriately, as they might compare intuitive aspects like the time two different unit action components will use to execute or the energy use of moving compared to that of zooming, for example. Doing so, (9) is no longer dependent on $\boldsymbol{\kappa}$, but just on a single κ_i, typically κ_1, and the given relations m_i.

$$C(\boldsymbol{a}_t) = \kappa_1 \cdot \sum\nolimits_i m_i \cdot x_{t,i} \quad ; \quad m_i = \frac{\kappa_i}{\kappa_1} \tag{10}$$

The problem occurring here is the theoretically unlimited range of κ_1 combined with its independency of $H(s_{t+1})$ in general. So just choosing the first cost-factor randomly or by sense of proportion would mostly lead to an imbalance between the terms of the sum in (8), that is the system is either almost exclusively influenced by the curve of $-H(s_{t+1})$ or just trying to reduce added costs without regarding the entropy based reward. While the user might be able to choose a sufficient value for κ_1 in a simple environment beforehand, he will surely fail when actions and rewards become more complex. So the only way, apart from gaining it empirically, is to learn a cost-factor fitting the entropy based reward's order of magnitude and then integrate it into the Reinforcement Learning process.

Consequently, we have to introduce a criterion for rating different κ_1 while naturally not integrating κ_1 into this criterion directly. Regarding our demands we are eager for a cost-factor leading the agent to acquire an entropy based reward as high as possible on the one hand, while causing as little cost as possible on the other. To be more precise, we do not seek to reduce the costs themselves but the mixed and weighted amount of the various cost generating unit actions a_{i_u}. Doing so, the κ_1 under testing has no direct influence on its own rating. It just affects the decision process of the Reinforcement Learning regarding (10), resulting in eventually different actions to be taken. So using a rating function

$$f(\kappa_1) = \frac{-H(s_{t+1})}{\sum_i x_{t,i} \cdot m_i} = \frac{-H(s_{t+1})}{\frac{C(\boldsymbol{a}_t)}{\kappa_1}} \tag{11}$$

such a change in behavior will affect the denominator directly and usually also the nominator, leading to a different rating for the appropriate κ_1.

While (11) is laid out for a single-step task, the intention of this paper requires expanding it to work with multi-step problems up to episode length T. Therefore using the information content's difference of consecutive views $\Delta(-H(s_{t+1})) = -H(s_{t+1}) + H(s_t)$, the equation is changed to

$$\tilde{f}(\kappa_1) = \frac{\sum\limits_{t=0}^{T-1} \Delta(-H(s_{t+1}))}{\sum\limits_{t=0}^{T-1} \sum\limits_{i=1}^{n} x_{t,i} \cdot m_i} = \frac{\sum\limits_{t=0}^{T-1} \Delta(-H(s_{t+1}))}{\sum\limits_{t=0}^{T-1} \frac{C(\boldsymbol{a}_t)}{\kappa_1}} = \mathrm{CBR} \quad . \tag{12}$$

At this point, it becomes obvious why to use a multi-step approach. Instead of calculating a conjoint, single sum of the nominator and denominator in (12) for each time step this is done separately since we would like to find one κ_1 that maximizes the nominator over an entire episode while minimizing the denominator over an entire episode at the same time. So actually we do not care about the ratings of every single time step, what a combined sum would imply. Furthermore, matching this demand we need to provide a multi-step approach for Reinforcement Learning with $\gamma > 0$, because otherwise calculating such a conjoint sum would always result in the same rating as in (12).

When finally applying each κ_1 to be tested to several episodes within the training phase of Reinforcement Learning, the one resulting in the highest values for (12) on average is regarded to be the optimal one. A further assumption made in our approach is the permanent existence of an action component $a_{t,n}$ in (1) related to taking views. This turns out to be quite useful since our demand always is to solve the task with the minimal amount of views if additional costs allow to do so. Moreover (12) will be called the CBR for cost-benefit ratio in the following due to its appearance.

4 Learning an appropriate cost-factor

So far, the two main issues of our viewpoint selection approach were described. On the one hand, it is necessary to learn a cost-factor κ_1, and on the other hand a regular improvement of the Q-base, influenced by the costs, is needed. Therefore, a consequential first thought is to just perform a whole training phase with a fixed κ_1 and to repeat this for all cost-factors one would like to examine. Then comparison could simply be done by using (12) in a global way, i.e. by calculating the average of all episodes' CBR. As this process would result in a maximal CBR for a certain κ_1, it is just a question of iterations to get closer to the global maximum. Although this would work for sure, the drawback is obvious since we would have to perform a multiple of training episodes, which becomes unbearable soon. For this reason, the following three subchapters will introduce mainly independent approaches for keeping the amount of necessary episodes within the training phase as small as possible.

4.1 Expanding the Q-base

So, a mostly parallel procedure for κ- and Reinforcement Learning had to be found while handling the occurring interdependencies at the same time. Regarding this, the main problem turned out to be the fact that a $Q(s', \boldsymbol{a}')$ learned with κ_{1x} usually does not provide a good strategy when using κ_{1y}. Thus already executed episodes might store information in the Q-base that is completely useless or even harmful for later evaluation if κ_1 is completely different. This consequently results in a lot of episodes executed for nothing and finally missing for providing a more reliable set of reference values.

To reduce this drawback in a first instance an expansion was added to all the $Q(s', \boldsymbol{a}')$ in the Q-base. While so far just storing state s_t, action $\boldsymbol{a}_t$ and $Q(s', \boldsymbol{a}')$ itself, additionally all actions $\boldsymbol{a}_{t+n}$ temporally following time t, all afflicted single-step entropy based rewards $-H(s_{t+n})$ and the κ_1 those steps were performed with earlier, will now be linked to and stored with their $Q(s', \boldsymbol{a}')$. Those elements L' at n time steps after following $\boldsymbol{a}'_t$ in s'_t are then accessible via

$$L'(Q(s'_t, \boldsymbol{a}'_t), t+n) = L'_{t+n} \quad \text{with} \quad L' \in \{\boldsymbol{a}, -H(s), \kappa_1\}. \tag{13}$$

Please note that regarding the convergence criterion of Reinforcement Learning, episodes' steps are performed randomly ($\epsilon = 0$) at the beginning of the training phase and get an increasingly higher probability ϵ of being performed greedily by applying (7) when training lasts. If now a time step is only followed by random actions, we can find out the stored, but actually not agent-influencing history information L'_{t+n} and replace it with the currently valid parameters, e.g. the current κ_1. Thus, random episodes, which mainly appear at the beginning of a training phase, can be used as a common database for all κ_1 to be tested. Of course, even greedy steps or whole episodes of an earlier evaluated κ_{1x} can serve as such a random base for each κ_{1y}.

4.2 Demands on parallel learning

Having a commonly usable base of random actions and ratings this way, the actual search for κ_1 can begin. To do so, an optimized search within a theoretically unlimited, but actually generously chosen, range of κ_1 is performed via an Adaptive Random Search algorithm [9]. But an enormous problem occurring here again refers to the interdependencies in κ- and Reinforcement Learning since the latter needs to have greedy steps as well as random steps within one episode. On the other hand κ-learning would prefer to rate only episodes performed completely greedily in order to achieve the true CBR. Since the demand is to parallelize the learning algorithms, a compromise has to be found. So what is done is that each first step in an episode is performed greedily, while each following step might be either greedy or random. This proceeding turns out to be acceptable, because first steps should be evaluated well enough within the preceding, all-random training phase, resulting in no drawback for the Reinforcement Learning.

If then an episode's ith step is eventually random we apply an originally Reinforcement Learning sub-strategy to the κ-learning, namely the Q-Learning [10], which is called an off-policy strategy but has been shown to converge to the results of the common on-policy calculation. The method is to just replace the rewards and costs that would have been gained if finishing the episode greedily, by those that are stored in the one $Q(s', \boldsymbol{a}')$ providing the most influential component when calculating $\widehat{Q}(s, \boldsymbol{a})$, if continuing greedily. This $Q(s', \boldsymbol{a}')$ will be called $\widetilde{Q}(s', \boldsymbol{a}')$ from now on. Since each $Q(s', \boldsymbol{a}')$ stores an array of experienced rewards and actions over time (see section 4.1), these rewards can now be accessed with the help of (13), leading to (14, 15) where the approximative calculations of the components in (12) are shown. Doing so forces the algorithm to calculate one extra action finding optimization step even though proceeding in a random way actually, but assures we do not lose the whole episode, i.e. not having performed all the episode's eventually preceeding optimization iterations for nothing and also gaining an at least approximative CBR.

$$\sum_{t=p}^{T-1} \mathit{\Delta}(-H(s_{t+1})) = \sum_{t=p}^{T-1} (\mathit{\Delta}(-H(s)))(\widetilde{Q}(s'_p, \pi(s'_p), t+1) \ \ ; \ \boldsymbol{a}_p \in \text{random} \tag{14}$$

$$\sum_{t=p}^{T-1} \frac{C(\boldsymbol{a}_t)}{\kappa_1} = \sum_{t=p}^{T-1} \frac{C(\boldsymbol{a})}{\kappa_1} (\widetilde{Q}(s'_p, \pi(s'_p), t) \ \ ; \ \boldsymbol{a}_p \in \text{random} \tag{15}$$

Obviously a rating for κ_1 becomes more reliable when step number p becomes higher since the approximation is over less steps, but it also depends on the quality of the Q-base growing continuously within a training phase.

4.3 Providing comparability

Given the Q-base learning supporting feature of complete randomness in choosing each episode's starting position, this constitutes another problem when trying to learn κ_1 simultaneously. So because of having to compare CBRs of episodes with generally different starting points, κ_{1x} might be disadvantaged to κ_{1y} right from the start if the initial position is worse in a global manner, i.e. if reaching best view points at a greater distance on average. Thus an objectively better κ_1 might be rated worse just because of the awkward initial position of its associated episode. Since we need to provide comparability, the character of an initial position has to influence κ-learning as well, leading to the definition of a state quality $Z(s_t)$ (16) and its integration to the calculation of CBR (17).

$$Z(s_t) = \frac{\int_{\boldsymbol{a}_t} \psi(s_t)\, d\boldsymbol{a}_t}{\max_{\boldsymbol{a}_t} \psi(s_t)} \quad ; \; \psi(s_t) = \frac{\sum_{t=p}^{T-1} (\Delta(-H(s)))(\widetilde{Q}(s'_p, \boldsymbol{a}'_p), t+1)}{\sum_{t=p}^{T-1} \frac{C(\boldsymbol{a})}{\kappa_1} (\widetilde{Q}(s'_p, \boldsymbol{a}'_p), t)} \tag{16}$$

$$\text{CBR}_{\text{new}} = \text{CBR} \cdot Z(s_0)^{-1} \tag{17}$$

By integrating over the expected long-run rating of all possible immediate actions $\boldsymbol{a}_t$ in current state s_t in (16) one calculates a state dependent rating, while the normalization in the denominator of $Z(s_t)$ assures we are able to compare even κ_1 learned with different object classes or between episodes occurring at larger distances in the training phase. Integration into the CBR then has to be done inversely since each κ_1 rated by an episode with a convenient starting position should be devaluated adequately and vice versa. It is worth annotating that there is no need or benefit in applying $Z(s_t)$ to any other states $s_{t>0}$ reached within an episode as those are not acquired randomly but are already part of the strategy followed with the actual κ_1.

Recalling section 2, fortunately most of the above mentioned work has already be done. This is because during the ARS-simplex optimization process (7) for finding the next best action concerning the Q-base, a huge amount of actions has to be evaluated anyway. Noticing that there is a global optimization first in the ARS-simplex, we can just take these global actions' results for calculating $Z(s_0)$ since the range of actions is sufficiently and almost equally covered, approximately representing the prior integral.

5 Experimental results

For showing the resulting effect of cost integration, a comprehensible viewpoint selection task was chosen, having to differentiate four kinds of similar cups with either letter A or B on the one side and numeral 1 or 2 at the opposing point (fig.2). The agent is just able to move on a circular line around the cup while each view within the range of $\pm$ 60 degrees from a character's center (90° resp.270°) could be reasonably taken for classification, but with decreasing reliability. When applying the introduced algorithms to this problem there is just one cost-factor ratio $\frac{\kappa_2}{\kappa_1}$ to be considered, where κ_1 is set to be the cost of moving the agent by one degree and κ_2 represents the cost of taking another view. The Reinforcement Learning technique was chosen to be Monte Carlo Learning with $\gamma = 0.2$ and a maximal episode length of five steps.

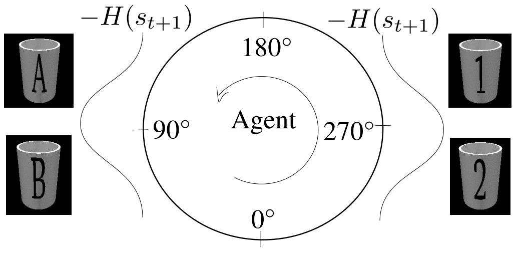

Fig. 2. Arrangement of the evaluated viewpoint selection task

Learning was then done with 800 training episodes for each cost-factor relation in tab.1, whereas the initial 200 episodes were performed exclusively at random for acquiring a reliable database. During the following 600 episodes the probability of greediness was chosen to increase in a linear way up to $\epsilon = 1$. Taking different ratios m_2^{-1}, tab.1 shows the actually learned κ_1, the averaged resulting classification rate, the resulting averaged amount of additionally taken views and the averaged performed move length within an episode. Concerning the classification rate, it has to be noticed that this environment was chosen to be very accommodating for keeping the side-effects small in order to bring out the decisive results more clearly. A pre-limitation of $\kappa_1 \in [0.0001; 0.1]$ was arranged.

m_2^{-1}	learned $\kappa_1 * 10^4$	classification rate [%]	views/episode	move[°]/episode
without costs	-	100	2.47	**197.12**
1:100000	692.83	100	2.47	198.02
1:10000	937.29	100	2.47	197.33
1:1000	350.23	100	2.48	195.49
1:100	17.23	100	2.51	190.80
1:80	3.30	100	2.55	181.37
1:60	9.46	100	2.59	165.67
1:50	4.71	100	2.66	**141.88**
1:40	24.13	89	3.02	112.82
1:30	24.76	41	5.17	30.55
1:10	28.39	39	5.60	7.96

Table 1. Resulting behavior for different cost-factor ratios

Regarding tab. 1, it becomes obvious that the system is able to change its behavior decisively when applying costs to the agent's actions. The results show that for this special environment we enabled the agent to save up to more than 50 degrees ($\sim 25\%$) of its rotational movement, compared to a former cost-insensitive learning, without reducing the classification rate. This benefit is due to the fact that the agent now does not reach for the absolute best view on each side of the cup any more, but founds its classification on a less reliable, but obviously still satisfying viewpoint if a gain in cost reduction is available this way. Since doing so is always afflicted with the risk of taking a needless view, this earning increasingly declines when taking views becomes more expensive.

Furthermore we even got the chance of influencing the system by small m_2 in a way that mostly prevents correct classification if this becomes too expensive, which also appears to be intuitive and meaningful. And without losing any prior abilities we can still simulate a cost-insensitive system by just taking extremely large values for m_2, as the first lines of tab.1 prove. This is due to reserving a cost component for taking views in combination with initiating a multi-step approach.

6 Summary and future work

In this paper it was shown how to expand an already existing Reinforcement Learning based viewpoint selection problem to a more foresighted version. Since the task of object classification could already be solved in an optimal way by taking a minimal amount of views, attention was towards also considering any kind of costs appearing at the same time, e.g. the agent's movement. Integration of those costs to the learning process was revealed, where costs need to be settled in an appropriate range in dependency of the environment's entropy based reward. To do so, a rating function was introduced for learning an appropriate single cost-factor the problem could be reduced to.

Therefore, the main attention was on not increasing the effort of pure, cost-insensitive learning, e.g. the amount of episodes, in a significant manner. So parallel strategies of the basic Reinforcement Learning and the newly introduced cost-factor learning were developed, regarding the reusability of already obtained values as well as coordinating the various differing demands of both techniques at steps within a training phase.

Upcoming work should now tend towards approaches for rating arbitrary cost-factors even by episodes performed with completely different cost-factors, and for eventually making use of completely random episodes for rating. The ability to provide a rating as often as possible is essential, since tasks with broadly higher dimensional action spaces call for a more founded and more reliable evaluation of the cost-factor in order to still find the optimum, while furthermore the cost-factor's reasonable range should become barely pre-limitable beforehand, leading to a larger search space.

References

1. Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, Massachusetts, 1995. Volumes 1 and 2.
2. F. Deinzer, J. Denzler, and H. Niemann. On Fusion of Multiple Views for Active Object Recognition. In B. Radig, editor, *Mustererkennung 2001*, pages 239–245, Heidelberg, September 2001. Springer.
3. F. Deinzer, J. Denzler, and H. Niemann. Viewpoint Selection – Planning Optimal Sequences of Views for Object Recognition. In N. Petkov and M. Westenberg, editors, *Computer Analysis of Images and Patterns – CAIP 2003*, LNCS 2756, pages 65–73, Heidelberg, August 2003. Springer.
4. C. Elkan. Cost-Sensitive Learning and Decision Making When Costs Are Unknown. In *Proceedings of the 17th International Conference on Machine Learning*, 2000.
5. K.S. Hong, K. Ikeuchi, and K.D. Gremban. Minimum Cost Aspect Classification: A Module of a Vision Algorithm Compiler. In *10th International Conference on Pattern Recognition*, pages 65–69, Atlantic City, USA, June 1990.
6. K. Klein and V. Sequeira. View Planning for the 3D Modelling of Real World Scenes. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 943–948, Takamatsu, Japan, 2000.
7. L. Paletta and A. Pinz. Active Object Recognition by View Integration and Reinforcement Learning. In *Robotics and Autonomous Systems, vol. 31*, pages 71–86, 2000.
8. R.S. Sutton and A.G. Barto. *Reinforcement Learning*. A Bradford Book, Cambridge, London, 1998.
9. A. Törn and A. Žilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 1987.
10. C. Watkins and P. Dayan. Q-learning. In *Machine Learning, 8(3/4)*, pages 279–292, 1992.