Application of Reinforcement Learning in a Real Environment Using an RBF Network

Sebastian Papierok\textsuperscript{1} and Anastasia Noglik\textsuperscript{2} and Josef Pauli\textsuperscript{2}

Abstract. The application of reinforcement learning algorithms in the context of robot behaviour learning is a poorly explored and very promising area of research. In the present work the learned strategy which resulted from simulation has been applied in a real world environment. To achieve good results in the real world it was necessary to build a simulation environment which mirrors the reality up to a practicable degree. We use a radial basis function network to approximate the action-value function. To enforce a robot to learn a desired behaviour a special online reward model has been developed. The approach reality-simulation-reality has been used to optimise the learning process in the simulation and apply the method in reality afterwards. Additionally the advantages and disadvantages of the application of the RBF-features over coarse coding with binary features have been examined.

1 Introduction

This article deals with a robot navigation problem in which an intelligent system should learn autonomously to navigate a mobile robot through a test track without collisions and in adequate time. For perception of the environment the robot’s infrared sensors are used whose data are very noisy. A similar problem was considered in a simulation in [6]. Environment perception could also be done e.g. visual-based [1].

To realise reinforcement learning [9] the popular Sarsa(\(\lambda\)) approach will be applied in our studies. The action-value function will be approximated with an RBF network [7] [5], which will be used to control the robot’s drive system. Because of the short training time and high accuracy of the RBF neural networks this method can be applied on real-time problems. The noisy sensor values will be converted here into a control-signal. This method doesn’t need any information about the position and orientation of the infrared sensors so that a high level of flexibility can be ensured for the complete system.

The simulation should stick to the reality as closely as possible in order to make this method applicable on the real robot at a later time. For these purposes recorded real sensor data will be used in the simulation in order to use the learned control network in reality without the need for any further adaptation steps (see also [4]). This approach is described in [8] as virtual prototyping. An advantage of this approach is that the implemented algorithm can be tested under ideal conditions to determine appropriate learning parameters and clear potential faults as well. Afterwards, the complete learning process will take place in the real world environment.

The applied fitness function for evolutionary algorithms from [2] was used as a basis for the development of an online reward model in this work. In [2] a control program for a similar problem at which a population-based EANT approach was applied was developed. There the EA approach was used to create a neural network that controls the robot’s drive system. However, the downside of this method is that the learning process is very complex in reality. As an example: for a single episode 100 robots and 100 runs become necessary.

Practical experiments show that robustness and learning ability of the robot perform better by applying behaviour learning with RBF networks in comparison to coarse coding approximation. The application of an online reward model shows promising results.

2 Background of Reinforcement Learning

2.1 The Reinforcement Learning Problem

In the field of reinforcement learning an agent should learn autonomously to achieve specific goals. The agent observes the current state of the environment and makes action decisions depending on the current state. As a reaction for the executed actions the agent receives the next state and a numerical reward. By means of the reward the agent can evaluate its action decisions and improve its behavior over time. An appropriate reward model is needed in order to ensure that the agent achieves the given goals. The agent’s objective is to maximize the sum of the rewards, also referred to as return, over the time. RL algorithms therefore try to estimate the expected return with the aid of so-called value functions.

2.2 Temporal Difference Learning

Temporal difference methods (TD methods) don’t need a predefined model of the environment. They learn only by the experience that the agent gains by interacting with the environment. Moreover the agent can use its new experiences immediately to improve its behaviour. For estimation of the value function the states and rewards that the agent observes while interacting with the environment are used:

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))
\]

By using continuous state spaces function approximators are used for estimating the value function based on samples that are observed during the interaction between agent and environment. Function approximators have a generalisation-ability so that the learning process occurs not only for one state-action pair but also for similar state-action pairs. The agent therefore must not visit all state-action pairs to make good action decisions. A function approximator can be considered as a function that is dependent on a real-valued parameter vector.

\textsuperscript{1} email: sebastian.papierok@stud.uni-due.de
\textsuperscript{2} Universität Duisburg-Essen, Lehrstuhl Intelligente Systeme, Germany, email: \{anastasia.noglik, josef.pauli\}@uni-due.de
At TD methods the agent can use his new experiences immediately. Therefore, a method is needed that allows an incremental adaptation of the approximated value function. For this purpose function approximators in conjunction with gradient-descent methods are used.

2.3 Linear Approximation of the Value Function

The approximated value function depends linearly on the parameter vector of the function approximator. The relation between the parameter vector and the real states will be described by so-called features that are collected in a feature vector. The features are distributed in the real state space and have a determined dimension and shape. Usually they are reciprocally arranged so that each possible state will be described by at least one feature.

When applying coarse coding the features can hold only the values 0 and 1 and are hence described as binary features. A feature has the value 1 if it represents the current state or 0 if not.

Radial basis functions are a generalisation of coarse coding. The co-domain of this features is located in the intervall [0,1]. An RBF-feature \( i \) can be defined as a Gauss error distribution curve that has a given width \( \sigma_i \) and position \( c_i \):

\[
\phi_s(i) = \exp\left( -\frac{\|s - c_i\|^2}{2\sigma_i^2} \right)
\]

3 Concept for Combining Reinforcement-Learning with an RBF Network

3.1 RBF Network for Approximating the Value Function

A separate parameter vector \( \vec{\theta}_{a,t} \) is used for each action \( a \). This vector has the same number of components as the feature vector \( \vec{\phi}_s \). The approximated action-value function is defined as:

\[
Q_t(s,a) = \vec{\theta}_a^T \vec{\phi}_s = \sum_{i=1}^{n} \theta_{a,i}(i) \phi_s(i)
\]

Figure 1 shows the RBF network that is used to approximate the action-value function.

3.2 Gradient-Descent Sarsa(\( \lambda \))

Figure 2 shows the gradient-descent Sarsa(\( \lambda \)) algorithm. The base algorithm of [9] was modified so that radial basis functions are used as features. Moreover it was adapted to the structure of the RBF network of figure 1.

The algorithm terminates if the specified number of episodes has been reached. An episode ends if the maximum number of steps has been reached or the agent has found the specified goal. At a collision of the robot with an obstacle the actual episode will be finished as well because the robot is only allowed to move forward. The problem is that the pivot point of the robot is located slightly in front so that a rotation is not sufficient to rescue it from this situation.

In the base algorithm of [9] the set \( \mathcal{F} \) contains the indices of the features that represent the current state-action pair \((s, a)\). In this work the set \( \mathcal{F}_s \) is only dependent on the current state \( s \). The action \( a \) is required once in the output layer of the RBF network. This approach has the advantage that the number of features is independent of the number of actions. Moreover the calculation effort can be reduced this way if the \( Q \)-value for only one state-action pair is required, which is the case when the agent chooses a random action.

01 Initialize \( \vec{\theta} \) arbitrarily
02 Repeat (for each episode):
03 \( \vec{e} = \vec{0} \)
04 \( s, a \leftarrow \) initial state and action of episode
05 \( \mathcal{F}_s \leftarrow \) set of features present in \( s \)
06 Repeat (for each step of episode):
07 For all \( i \in \mathcal{F}_s \):
08 \( e_{a,i}(i) \leftarrow e_{a,i}(i) + \phi_s(i) \)
09 Take action \( a \), observe reward, \( r \), and next state, \( s' \)
10 \( \delta \leftarrow r - \sum_{i \in \mathcal{F}_s} \theta_{a,i}(i) \phi_s(i) \)
11 With probability \( 1 - \epsilon \):
12 For all \( a' \in \mathcal{A}(s') \):
13 \( \mathcal{F}_{s'} \leftarrow \) set of features present in \( s' \)
14 \( Q_{a'} \leftarrow \sum_{i \in \mathcal{F}_{s'}} \theta_{a',i}(i) \phi_{s'}(i) \)
15 \( a' \leftarrow \arg \max_{a} Q_{a} \)
16 else \( a' \leftarrow \) a random action \( \in \mathcal{A}(s) \)
17 \( \mathcal{F}_{s'} \leftarrow \) set of features present in \( s' \)
18 \( Q_{a'} \leftarrow \sum_{i \in \mathcal{F}_{s'}} \theta_{a',i}(i) \phi_{s'}(i) \)
19 \( \delta \leftarrow \delta + \gamma Q_{a'} \)
20 \( \vec{\theta} \leftarrow \vec{\theta} + \alpha \vec{e} \)
21 \( \vec{e} \leftarrow \gamma \lambda \vec{e} \)
22 \( s \leftarrow s' \)
23 \( a \leftarrow a' \)
24 until \( s \) is terminal

Figure 1. RBF network for the approximation of the action-value function.

Figure 2. Linear, gradient-descent Sarsa(\( \lambda \)) with RBF-features and \( \epsilon \)-greedy policy.
3.3 State Space

For perception of the environment the infrared sensors of the robot are used. The state space is defined as \([[10, 65] \cup \{-1\}]^7\). The interval \([10, 65]\) specifies the used measurement range of the infrared sensors. The value \(-1\) is used if the sensors does not detect any obstacle. A state is composed of the measured values of seven infrared sensors. There are three RBF features per dimension of the state space so that the 7-dimensional state space has a total of \(3^7\) features. Figure 3 shows the distribution of the features for one sensor.

![Figure 3. Distribution of the radial basis functions for one infrared sensor.](image)

Figure 3. Distribution of the radial basis functions for one infrared sensor.

Again, the sensor values are very noisy. At an ideal distance of 45 cm the sensors supply values between 35 cm and 70 cm [3]. Moreover there are sensor failures at which the obstacles will not be detected at all. Figure 4 shows the simulated infrared sensors of the robot.

![Figure 4. Simulated infrared sensors of the robot.](image)

Figure 4. Simulated infrared sensors of the robot.

3.4 Action Set

The action set consists of three actions: turn right, move forward, and turn left. An action is defined as a tuple \((\Delta \varphi, \Delta s)\) at which \(\Delta \varphi\) describes an angle of rotation and \(\Delta s\) a stroke for moving forward. It is assumed that the absolute orientation and position of the robot are unknown so that an action describes only the change of orientation and position respectively. The forward and rotational velocity are assumed as constant. For the reward model the actions are defined as:

\[
\alpha \in \{-1, 0, 1\} \quad \text{with:} \quad 
\begin{align*}
-1 & \quad \text{turn right} \\
0 & \quad \text{move forward} \\
1 & \quad \text{turn left}
\end{align*}
\]

4 Online Reward Model

The intention at the development of a reward model is to provide the robot a specific behaviour in specific situations under the additional condition that the number of episodes should be as low as possible. A similar problem was treated in [4] (by our working group) with evolutionary algorithms (EA) and was successfully used in reality.

In the following the similarities between the evolutionary approach and the reinforcement learning approach will be discussed. Candidate solutions to the optimisation problem play the role of individuals in a population (for example in [2]) where an individual is the artificial neural network that controls the robot. The fitness function determines the learning process of the individual. Individuals in EA are with \(Q_{app}^\pi\) defined policies (here the RBF networks) that are comparable with the RL approach. The fitness function of the individual is almost comparable with the approximation of the value function \(Q_{app}^\pi\) for policy \(\pi\).

Because of this similarity the elementary reward of the developed reward model is based on the fitness function from [2]. That fitness function favors the desired behaviour of the robot as well. The fitness function is defined as \(F = \sum_{t=1}^{T} f(t)\), where \(f(t) = v(t) \exp(-100(H(t) - H(t-1))^2)\) where \(s_{min}(t)\) is the according value at time \(t\). \(v(t), H(t)\) and \(s_{min}(t)\) are the speed, the heading of the robot, and the minimal value of the sensor set readings respectively.

Observations have shown that the penalty mechanism at the evolutionary approach is built in but not easy to find. This is relevant, if the individual stands still due to a collision and will not be rewarded further. The population orientated method through several possible solutions is leading quickly to the desired solution.

In the reward model that is used in this work the penalty mechanism will be introduced artificially. The minimal sensor value of the sensor set is defined as \(s_{min} := min_{i,s_i(t)}\). The penalty mechanism will be activated if the minimal sensor value falls below a critical threshold \(s_{min}\). The value \(s_{min}\) depends on the design of the robot, its balance point, and its dimensions. With the help of the \(a_{desired}\) value a kind of reflex will be introduced that acts rational. In case the robot receives a message from the right side about a shortfall of the minimal acceptable distance \(s_{min}\) the rationality says: turn right and move forward is not the correct decision so that it applies to \(a_{desired} = 1\). The negative elementary reward will be computed depending on \((s_{min} - s_{min})\) and the value \([a_{desired} - a_t]\).

The complete reward model is defined as:

\[
\begin{align*}
R_s(s_{min}, a_t, a_{t+1}) = \\
\begin{cases}
-1 - (s_{min} - s_{min}) |a_{desired} - a_t| + 1 & : \text{for negative reward} \\
(s_{min} - s_{min}) \cdot \exp\left(-\frac{(a_t - a_{desired})^2}{2\sigma^2}\right) & : \text{for positive reward}
\end{cases}
\end{align*}
\]

where \(\sigma = 0.75\). The elementary reward is hence dependent on the current state \(s_t\) as well as the two last executed actions \(a_t\) and \(a_{t+1}\). It concerns a reflex that protects the agent’s life. It is important to find a balance between rewards and penalties. The reward for goal reaching should at least differ by a factor of 10.

5 Results and Discussion

There were several tests performed to determine the robustness and the learning ability of the RBF network which controls the robot’s
drive system. The evaluations have shown that the used RBF network has the ability to process the sensor data so that interferences like sensor failures or noisy sensor values have no strong influences on the learning process and the application of the method. It could be shown that the simulated robot can navigate through the given test track after few episodes (after one episode as well) without collisions. The knowledge that was learned in the simulation was applied in the real world without additional effort so that the robot was able to navigate through the real test track without collisions as well.

5.1 Comparison between RBF and Coarse Coding

In the following the characteristics of the intelligent system such as robustness and efficiency will be determined by applying of radial basis functions and coarse coding.

The robustness and efficiency of the method will be considered in the Grid World. This is defined as an extension of the Grid World [9] to a continuous state space which is represented by the position and orientation of the robot. Feedback from the environment in Map World is the robot’s position and orientation in the world coordinate system. The start and goal positions are fixed. This environment was chosen to better understand the learning process of the implemented algorithm. The behaviour of the robot is statistically not easy to interpret. In contrast, the number of steps to a fixed goal is always ascertainable. Consequently, the statements about the robustness and efficiency of the learning process can be derived easier. Moreover, the required time to solve the task in the Map World is much less compared to the measuring space. As a result the number of trials and the significance of the results can be increased. It is assumed that the characteristics of the method are portable to other state spaces as well.

5.1.1 Test Method

To compare RBF and coarse coding the position and orientation of the robot are used for perception of the environment so that the agent optimises the path from the start position to the specified goal. The agent receives a positive reward if it finds the goal. In the case of a collision of the robot with an obstacle the agent gets a penalty (negative reward). In the other cases it receive a small penalty. To understand the learning process an additional episode will be passed between each second episode at which the learning rate and the exploration rate are both set to the value 0. It will be described hereafter as sample. A sample describes the number of steps that the simulated robot needs from the start position to the given goal. After the additional episode the learning process will be continued.

The following diagrams show the average values of the samples from 200 trials at which the simulated robot has reached the goal. To compute the number of samples that were used for the computation of a specific average value the probabilities for goal reaching are shown in the diagrams as well.

5.1.2 Different Exploration Rates

In the following the convergence properties of the method using different exploration rates \( \epsilon \) will be analysed.

At a sensor failure the current state will be described by another subspace of the state space. This means that the agent can choose an action that is possibly inapplicable in the present situation. At a large measurement difference of a sensor the agent can choose a wrong action as well because the current state will be possibly represented by other features. Such behaviour of the agent is comparable with the selection of random actions.

Figure 6 shows the results for \( \epsilon = 0.05 \) using environment 1. The method provides good results for both coarse coding and radial basis functions. The optimal path was found after 130 episodes for both types of features.

![Figure 5. Start configuration of the test environment (environment 1). The arrow describes the robot’s driving direction.](image-url)

![Figure 6. Comparison between RBF and coarse coding using a small exploration rate \( \epsilon \). Test configuration: \( \alpha = 0.2/|F_s| \), \( \epsilon = 0.05 \), \( \gamma = 0.8 \), \( \lambda = 0.8 \), \( \sigma = 7 \).](image-url)

Figure 7 shows the results for \( \epsilon = 0.6 \). It is noticeable that the method is susceptible to interferences by using coarse coding while it is stable and converges towards a specific value after 240 episodes using radial basis functions. It is noticeable that the probability for goal reaching by using coarse coding no longer achieves the value...
1 but takes course highly unstable at smaller values. Moreover the optimised path by using radial basis functions is a bit better than in figure 6.

Figure 7. Comparison between RBF and coarse coding using a high exploration rate $\epsilon$. Test configuration: $\alpha = 0.2/|\mathcal{F}_s|$, $\epsilon = 0.6$, $\gamma = 0.8$, $\lambda = 0.8$, $\sigma = 7$.

Binary features also have heavy effects if the current state is located far away from the centre of a feature so that the computed linear combination contains components of the parameter vector that are not important for the representation of the current state as well. At small exploration rates state-action pairs already known by the agent will be visited again and again so that the agent does not differ from its current strategy. At high exploration rates the agent visits unknown state-action pairs time after time. It is believed that the agent forgets its gathered knowledge if it reaches a state-action pair at which it must discover the goal again. This effect will be enforced as a result of long eligibility traces because the before attended state-action pairs will be adapted over a longer period of time.

The described effect can not occur when using radial basis functions because the components of the parameter vector are included only scaled in the computation of the approximated action-value function.

5.1.3 Different Dimensions of the Features

In the following several tests are performed to determine the robustness of the method when using different dimensions of the features. For this reason the width of the features will be modified i.e. support of Gaussian. It is assumed that the features are distributed in the state space evenly so that each state can be described at least by one feature. To analyse the results independently of the position and dimension of the features the minimum and maximum number of overlaps will be determined. The exploration rate will be set to the value 0.2 to simulate a small interference.

Figure 8 shows the results for $\sigma = 7$. It can be noticed that the method provides a good result for both coarse coding and radial basis functions.

The results for $\sigma = 11$ are shown in figure 9. By using coarse coding the same effect can be observed here as in figure 7.

By using coarse coding the difference $Q_{CC}(s,a) - Q_{RBF}(s,a)$ will increase with the growing number of overlaps because the com-

Figure 8. Comparison between RBF and coarse coding using a small number of feature overlaps. Test configuration: $\alpha = 0.2/|\mathcal{F}_s|$, $\epsilon = 0.2$, $\gamma = 0.8$, $\lambda = 0.8$, $\sigma = 7$.

Figure 9. Comparison between RBF and coarse coding using a high number of feature overlaps. Test configuration: $\alpha = 0.2/|\mathcal{F}_s|$, $\epsilon = 0.2$, $\gamma = 0.8$, $\lambda = 0.8$, $\sigma = 11$. 

computed linear combinations have more and more components of the parameter vector that are not very important for the representation of the current state. This means that the number of features is growing in which the current state is located far from the centre.

By using RBF features the method converges towards a specific value, but the number of required episodes is higher than in figure 8. This is because the adaptation of the parameter vector requires more time in conjunction with a growing number of overlaps, because the sensibility of the computed linear combination is growing already at small state changes. Therefore, a high number of overlaps has negative effects on the method.

The method provides good results if each possible state is represented at least by two features. The number of overlaps must consequently not be that high in order to use this method efficiently and robustly.

5.2 Learning of Behaviour in Online Mode

As stated before, the agent should learn to navigate the robot through a given test track without collisions and in adequate time. In the first instance the process pattern reality-simulation-reality was used. In this case, the learning process took place in the simulation.

Several reward models have been tested in the simulation. The intelligent system uses an online reward model with which the agent can improve his behaviour while it is navigating the robot through the test track. The agent gathers its knowledge because of the states, actions, and rewards that occur during the interaction with the environment. There is a test track given at which the agent has to learn the correct behaviour without previous knowledge.

Observations have shown that the robot was already able to navigate through the test track after a few episodes. A collision can happen once in a while if several sensor failures or wrong sensor values respectively occur in a row. To reduce the influences of interferences the forward velocity of the robot can be decreased. Afterwards the knowledge that was learned in the simulation was applied on the real robot. In reality the robot could navigate through the test track as well.

The learning rate \( \alpha \) is set to the value 0.2 during the learning process. The exploration rate \( \varepsilon \) is adjusted to the value 0.01 in order that the agent shall generally use its gathered knowledge to navigate the robot through the test track. The discount factor \( \gamma \) and the decay factor \( \lambda \) are both set to the value 0.8.

In the second instance the complete learning process has been performed in reality (see figure 10). The agent has to learn the correct behaviour without previous knowledge here. Observations have shown that few episodes are needed in order that the agent can navigate the robot through the test track without collisions. The learning process in the reality has been recorded on video.\(^3\)

The intelligent system has the ability of generalisation so that the agent can learn its behaviour in a complex test track and use it in yet unknown test tracks.

6 Conclusion

The developed method offers two advantages. It is stable and leads quickly to the desired behaviour of the robot in reality. The used approach is the Sarsa(\( \lambda \)) algorithm from the field of reinforcement learning. Because of the short learning time and the adaptation ability RBF-features have been used for approximation of the action-value function. The RBF network creates a control signal from the measured sensor values so that the desired behaviour of the robot can be determined. It was necessary to develop an online reward model. The fitness function of evolutionary algorithms was applied as the base of the online reward model. The adaptation ability of the RBF network was confirmed by comparing it with coarse coding in the benchmark environment Map World. For the complete method the virtual prototyping approach was used. Numerous trials have shown that the approach is fast, efficient, and adaptive on difficult real environments as well.

REFERENCES


\(^3\) see http://www.uni-due.de/is/projekt_emrobnav.php