Real-world reinforcement learning for autonomous humanoid robot docking[☆]

Nicolás Navarro-Guerrero*, Cornelius Weber, Pascal Schroeter, Stefan Wermter

Knowledge Technology Group, University of Hamburg, Department of Informatics, Vogt-Koelln-Str. 30, 22527 Hamburg, Germany

Abstract

Reinforcement learning (RL) is a biologically supported learning paradigm, which allows an agent to learn through experience acquired by interaction with its environment. Its potential to learn complex action sequences has been proven for a variety of problems, such as navigation tasks. However, the interactive randomized exploration of the state space, common in reinforcement learning, makes it difficult to be used in real-world scenarios. In this work we describe a novel real-world reinforcement learning method. It uses a supervised reinforcement learning approach combined with Gaussian distributed state activation. We successfully tested this method in two real scenarios of humanoid robot navigation: first, backward movements for docking at a charging station and second, forward movements to prepare grasping. Our approach reduces the required learning steps by more than an order of magnitude, and it is robust and easy to be integrated into conventional RL techniques.

Keywords:
Reinforcement learning, SARSA, Humanoid robots, Autonomous docking, Real-world

[☆]Portions of this work were presented at the Annual Conference Towards Autonomous Robotic Systems (TAROS), Aug. 31 - Sep. 2, 2011.

*Corresponding author Tel.: +49 40 428 83 2530
Email address: navarro@informatik.uni-hamburg.de (Nicolás Navarro-Guerrero)

1. Introduction

Reinforcement learning (RL) [1, 2] is a biologically inspired learning paradigm consistent with the trial-and-error learning process related to the dopaminergic system [3]. RL models are able to learn through experience acquired by an agent interacting with its environment that tries to maximize rewards and minimize punishments. Given a certain initial condition, RL algorithms are particularly suitable for solving action sequences by building a value function that encodes the effect of different decisions.

For tasks with delayed reward, methods based on temporal-difference (TD) learning have been broadly accepted because of their simplicity requiring minimal computational power, as indicated by Sutton and Barto [1] and supported by a vast body of research [4, 5, 6, 7, 8, 9]. TD based methods do not require detailed models of the environment and are fully incremental, i.e. are capable of learning based on previous knowledge [1].

Reinforcement learning usually consists of many trials that begin with the agent's random initialization followed by many executed actions until the agent eventually reaches the goal. Following a few successful trials the agent learns beneficial state-action pairs based on its acquired knowledge. Learning is carried out using positive and negative feedback during the interaction with the environment in a trial and error fashion. In contrast with supervised and unsupervised learning, reinforcement learning may not use feedback for intermediate steps, but a reward (or punishment) may be given only after a learning trial has been finished. The reward is a scalar and indicates whether the result was right or wrong (binary) or how right or wrong it was (real value). This limited feedback characteristic makes it a relatively slow learning mechanism, but attractive due to its potential to learn action sequences that are not known by a teacher.

Researchers use RL broadly within simulated environments or for abstract problems [10, 4, 5, 6]. Here, a model of the agent-environment dynamics is available, which is not always available or easy to infer in real-world problems. Moreover, a number of assumptions, which are not always realistic, are made, e.g. on the state-action transition model, the design of the reward criterion, and the magnitude and kind of noise if any, etc.

On the other hand, real-world RL approaches are scarce [7, 8, 9], mostly because RL is expensive in data or learning steps, the state space tends to be large and the turnaround times for results are long. Moreover, real-world problems present additional challenges, such as safety considerations,

real-time action execution, changing sensor characteristics, actuators and environmental conditions, among many others.

Several techniques exist to improve real-world learning capabilities of RL algorithms. *Dense reward functions* [7] provide performance information for intermediate steps, thereby shaping the policy and restricting the emergence of novel unforeseen policies. *State space reduction* [7] is dependent on the particular problem and can be a very time-consuming designing task. Another approach proposes modification of the agent's properties to fit the given problem [8], which relies on a smart definition of the state space that accounts for a reduction of dimensionality. *Batch reinforcement learning* [9] uses information from past state transitions, instead of only the last transition, to calculate the prediction error function based on storage and reuse of state-action pairs. *Supervised reinforcement learning* [7, 6] is based on batch RL, but differs in the generation of training examples. In batch RL the state-action pairs are generated autonomously through random exploration while supervised RL uses human-guided action sequences during initial learning stages avoiding the costly random exploration.

The proven value of RL techniques for navigation tasks [11, 7] motivates us to develop a real-world RL approach for a humanoid robot to navigate autonomously into a docking station. This approach makes use of a supervised RL algorithm and Gaussian distributed state activation. We successfully tested this method in a simulated 2-dimensional grid-world and two real scenarios. The latter are a backward configuration for an experimental docking station for recharging and a forward docking configuration for grasping tasks. Our approach works with a reduced number of training examples, has few model assumptions, and it is robust and easy to incorporate into conventional RL techniques based on TD learning.

The paper is organized as follows. Section 2 presents the motivation and design criterion of the model. Section 3 presents the neural architecture, algorithm and training procedure. Section 4 demonstrates the proposed method in a simulated scenario. Section 5 presents the results of applying the method to two real-world applications. Section 6 analyzes comparatively both real-world scenarios. Section 7 presents conclusions.

2. Motivation and system design

Despite advances in humanoid robot control there are still difficulties in accurate maneuvering tasks, which we consider a key building block for do-

mestic applications of robots. In particular, humanoid robots, like the NAO [12], are used in a growing number of social, service and entertainment robot scenarios [13, 14]. One of the NAO's major limitations for domestic applications is the difficulty of precise localization, due to slippage and accumulated localization errors, which increases the difficulty to perform other tasks such as object grasping and delivering. Our interest of studying domestic robot applications motivates us to tackle the issue of precise localization and positioning. Here, we describe the successful docking of a NAO robot based on RL techniques, described in detail in Section 3. The two scenarios are both easy to build and "non-invasive", i.e. do not require major interventions on the robot's hardware and do not affect the robot's mobility or sensor capabilities.

2.1. Backward docking station for autonomous recharging

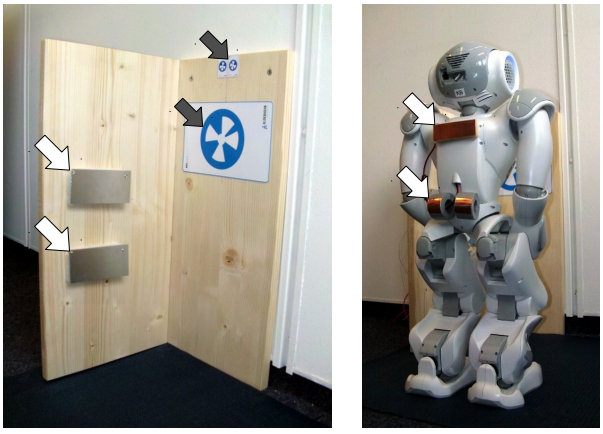
One of the NAO's limitations is its energetic autonomy, which typically does not surpass 45 minutes. This motivates the development of strategies to increase the robot's operational time minimizing human intervention [15]. Despite the challenge to maneuver the robot backwards, we chose to test a partial backward docking [15]. This offers advantages such as easy mounting on the NAO. It does not limit the robot's mobility, does not obstruct any sensor, nor does it require cables going to the robot's extremities, and allows a quick deployment after the recharging has finished or if the robot is asked to do some urgent tasks.

The prototype built for the proposed autonomous recharging is shown in Fig. 1(a). The large landmark (naomark[1]) is used for a hard-coded approaching behavior when the robot is more than 40 cm away from the docking station[2], while the two smaller landmarks are used for an accurate docking behavior for which we use the RL algorithm described in Section 3.

The overall autonomous recharging was split into four phases. During the first phase a coarse approach behavior takes place. This behavior is temporarily a hard-coded algorithm that searches for the charging station via a scanning head rotation followed by a robot rotation. The robot estimates the charging station's relative position based on geometrical properties of the large landmark and moves towards the charging station. We are currently

[1] 2-dimensional landmark provided by Aldebaran-Robotics

[2] Distance measured from the landmark to the robot's camera



(a) (b)

Figure 1: Backward docking station for NAO. (a) White arrows indicate the electrical contacts placed on the docking station and gray arrows indicate the landmarks' position. (b) Robot's electrical connections.

developing a more sophisticated approach in the KSERA project framework [14, 16] where we use a ceiling camera to locate the robot anywhere within an indoor room and to navigate the robot to a distance of approximately 40 cm away from the landmarks; see Fig. 2(a). In the second phase the robot re-estimates its position and places itself so that its left shoulder as well as its face are oriented towards the landmark, as shown in Fig. 2(b). In the third phase the RL algorithm is applied to navigate the robot backwards very close to the electric contacts as presented in Fig. 2(c).[3] After reaching the final rewarded position, in the fourth and final phase, a hard-coded algorithm moves the robot to a crouch pose; see Fig. 2(d). Then, the motors are deactivated and the recharging process starts.

2.2. Forward docking station for grasping

The second scenario aims at robot docking to allow autonomous grasping. We developed a similar docking structure, i.e. one big landmark and two small landmarks. A conceptual schema of the setup for grasping is depicted in Fig. 3. The forward docking consists of two phases. The first phase contains all the features of the coarse approach described for the backward docking.

[3]In this docking phase, NAO's gaze direction is oriented towards the landmark.

(a) Approach (b) Alignment (c) Docking (d) Crouch pose

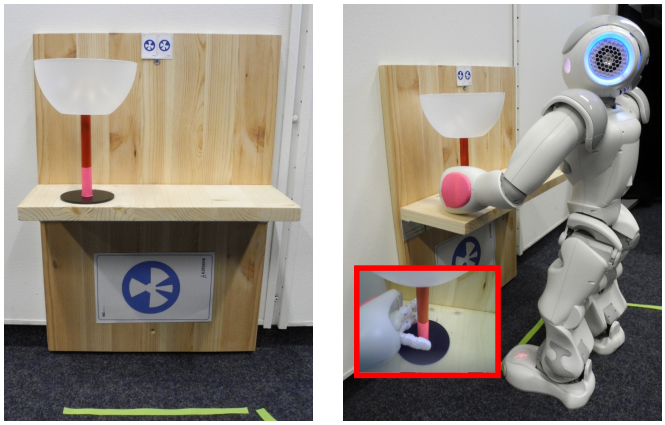
Figure 2: Top view of the autonomous robot behavior in its four different phases: (a) Approaching, (b) Alignment, (c) Docking and (d) Recharging in crouch pose.

Since the coarse approach behavior places the robot facing the landmarks at approx. 40 cm away, the transition from coarse docking to precise docking does not require an alignment phase. The second phase corresponds to the precise docking behavior implemented using the RL algorithm described in Section 3, which navigates the robot forward to the docking station and places it in 15 cm proximity of the landmark. Once the robot is in this position the grasping task is going to take place.

2.3. Choice of the learning system

In order to make reinforcement learning feasible in real-world scenarios several techniques had been developed, as presented in Section 1. From these techniques, supervised reinforcement learning [7, 6] offers the possibility of reducing the number of learning steps by avoiding the initial exploration of the state space. This is achieved by providing the agent with a few correct training examples and using them for off-line training.

We create the training examples by tele-operating the robot from several random positions to the goal position, while saving state, action and reward information. The off-line training consists of the presentation of the saved action and state vectors (or action sequences) to the agent. Thus, the agent can learn the given action sequences without additional real-world execution of actions. Since the training examples represent only a reduced subset of possible solutions, we use additional reinforcement learning to safely operate the robot around the near-optimal solutions provided by the operator. Particularly, we use SARSA learning, which is a classical on-policy algorithm



(a) (b)

Figure 3: Scenario for grasping a cup from a shelf. (a) Shelf with landmarks for accurate docking behavior and a graspable object. (b) NAO robot is in grasping position (the inset shows robot's view).

for TD-Learning. SARSA does not have major restrictions of convergence, and it can easily be combined with eligibility trace, opposed to Q-learning [1]. The mathematical implementation is detailed in Section 3.

In order to limit even more random exploration and to achieve efficient real-world reinforcement learning, we introduce an additional modification that boosts the learning speed. Instead of using a single active state at a given time, as conventionally used in reinforcement learning techniques, we use a Gaussian activation of state units [17]: a Gaussian is centered around the current robot state; see Fig. 4.

One motivation for a Gaussian state activation is that states close to the current state should often generate the same action. Using this concept, we can extend and spread what we know about a state to *neighboring* regions of the state space. This differs from eligibility traces that allow faster on-line learning by strengthening states *recently* visited. Repetitive off-line training, though, incorporates the effect of eligibility traces.

3. Model architecture and learning

The model has an input layer, which represents the agent's current state, and an output layer, which represents the chosen action. Both layers are fully connected (see Fig. 5). The number of states, actions and the size of the

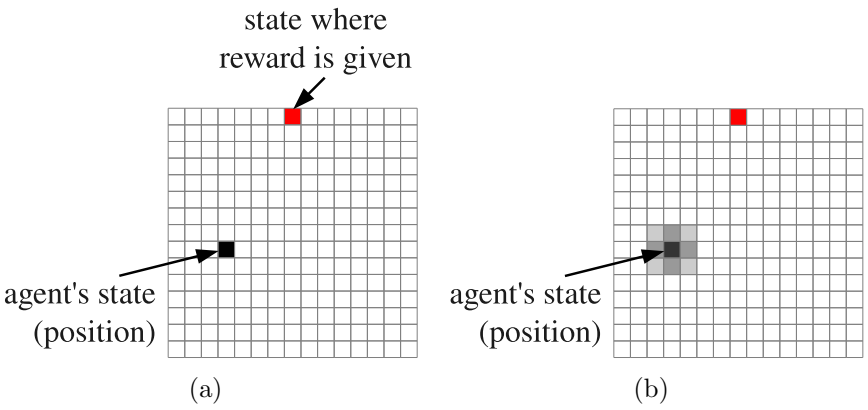


Figure 4: 2-dimensional grid-world example with two forms of state representation. The goal position is indicated by a red cell. Gray-scale indicates state activation. Left, single state activation at the agent's position. Right, Gaussian distributed state activation. The spread of activation to neighboring states speeds up learning.

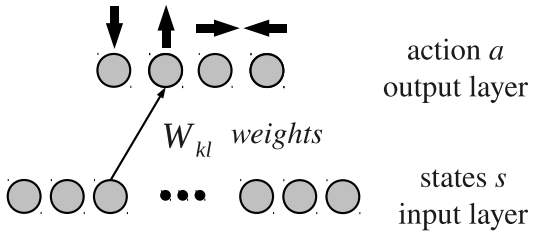


Figure 5: Neural network schematic overview. For clarity, only one connection weight is shown (thin arrow).

actions are adjusted empirically as a trade-off between speed and accuracy for each of the tested docking behaviors. The algorithm implementation will be explained using a grid-world example, which offers an intuitive ground and facilitates graphical representation of the modifications that are being introduced.

The navigation problem is modeled as a Markov decision process (MDP). An MDP is defined by a set of states S, a set of actions A, a transition model $P(s'|s,a)$ that specifies the probability of reaching the next state s' by taking action a in state s, a reward model $R(s',s,a)$ that specifies the immediate reward received when taking action a in state s, and an exploration policy $\pi(s|a)$, which is a mapping from states to actions.

Considering the 2-dimensional grid-world example shown in Fig. 4(a), the

state space S is formed by all cells. The goal position is indicated by a red cell and the current agent's position by a black cell. The agent's objective is to reach the rewarded goal position as quickly as possible.

The actions are moving `UP`, `DOWN`, `LEFT` and `RIGHT`. A move does not depend on the history but only on the policy $\pi(s|a)$, which depends on the learnt network weights W. A binary reward r is used to indicate whether the agent has succeeded or not. The agent is given $r = 0$ as long as the desired position is not reached. Once the goal position is reached, the agent receives $r = 1$ and the "trial" is finished.

The learning algorithm is based on SARSA [1, 2] and can be summarized as follows. For each trial the robot is placed at an initial random position within the defined workspace. The agent reads the cell's coordinates to obtain the internal state activation vector s, with all entries zero except for the entry that corresponds to the world position.

The net activation h_i of action unit i is computed as

$$h_i = \sum_l W_{il} s_l \quad , \tag{1}$$

where W_{il} is the connection weight between action unit i and state unit l. For the particular case that only one state unit l^* is activated, Eq. 1 becomes

$$h_i = W_{il^*} \, s_{l^*} = W_{il^*} \tag{2}$$

Connection weights W_{il} are initially set to zero. Next, we used a softmax-based stochastic action selection policy

$$P_\beta(a_i = 1) = \frac{e^{\beta h_i}}{\sum_k e^{\beta h_k}} \quad , \tag{3}$$

where β controls how deterministic the action selection is. Large β implies a more deterministic action selection or a greedy policy. Small β encourages the exploration of new solutions. We use $\beta = 500$ to prefer exploitation of known routes and to conservatively explore unforeseen policies. Based on the active state (l^*) and on the current selected action (k^*), i.e. $a_{k^*} = 1$; $a_{i \neq k^*} = 0$, the current estimate value $Q(s, a)$ is computed:

$$Q(s, a) = \sum_{k,l} W_{kl} a_k s_l \tag{4}$$

For the particular case of SARSA, where only one single state l^* and one action k^* can be active at a time, this becomes

$$Q(s,a) = W_{k^*l^*} a_{k^*} s_{l^*} = W_{k^*l^*} \tag{5}$$

The old state-action value $Q(s,a)$ is subtracted from the time-discounted new value $\gamma Q(s',a')$ to yield the network prediction error δ. The time-discount factor $\gamma \in [0,1]$ controls the importance of proximal rewards against distal rewards. Small values are used to prioritize proximal rewards. In contrast, values close to one are used to equally consider all rewards. Considering also the binary reward $r \in \{0,1\}$, the prediction error is computed as

$$\delta = \begin{cases} \gamma Q(s',a') - Q(s,a), & \text{if } r = 0, \\ r - Q(s,a), & \text{if } r = 1, \end{cases} \tag{6}$$

Eq. 6 differs from $\delta = \gamma Q(s',a') + r - Q(s,a)$, as presented by Sutton and Barto [1], from which the name SARSA originates. Our modified version works well with binary reward schemas and prevents unlimited growth of weight values. We set $\gamma = 0.65$. The weights are updated using a δ-modulated Hebbian rule with learning rate $\epsilon = 0.5$:

$$\Delta W_{il} = \epsilon \delta a_i s_l \tag{7}$$

At this point, the two techniques, introduced in Section 2.3, to facilitate real-world RL come into play. First, to avoid random exploration, a set of training examples are recorded and used for off-line training. Within each trial, the learning algorithm was realized as described in Eq. (1)-(7). However, instead of using Eq. (3) for stochastic action selection, the selected action was provided by the tele-operation data. We refer to this procedure as *"supervised reinforcement learning"*. Second, instead of using single state activation as in Eq. (2), where only a single input neuron has maximal activation ($s_l = 1$) at a time, we use a Gaussian activation of state units [17]: a Gaussian is centered at the current robot state (*"active state"*)

$$s_l = N \cdot e^{-\frac{(x_l - \mu_x)^2 + (y_l - \mu_y)^2}{2\sigma^2}} , \tag{8}$$

where N is a normalizing factor, i.e. the sum over the state space activations is 1. The different paradigms of state activation are shown in Fig. 4.

We use $\sigma = 0.85$, which effectively "blurs" the activation around the *"active state"*. In this way generalization to states that have not been directly visited is possible. The dimensionality of the Gaussian distribution will depend on the number of variables used to build the state space. In this grid world example, we show schematically a 2-dimensional Gaussian distribution. μ_x represents the current x-cell coordinate and μ_y the y-cell coordinate of the agent.

4. Analysis of results from simulation (grid-world)

In Table 1, we compare the performance of two supervised RL methods after off-line training, i.e. using *"single active state"* and using Gaussian distributed state activation. The training examples for supervised RL in both cases consist of 3 user-generated action sequences. The trajectories for the training examples include the borders and the central path and cover 15.2% of the state space. Testing was performed with 100 trials with random starting positions.

Results are shown after 300 off-line training trials, i.e. each of the 3 tele-operated example trials is repeated approx. 10 times. This number was sufficient for good performance. Training is governed by tele-operated policy π_{sup} without random exploration, i.e. without autonomous *"interaction"* with the environment. This would be appropriate to do with real-world hardware that must not run unattended. Testing is *"interactive"*, i.e. the agent action selection is governed by the learnt policy π_{sup^*}.

After training using single state activation, the agent's actions remain random in those states that have not yet been visited. This leads to a high STD of the number of steps required (see Table 1). After training with Gaussian state activation the agent generalizes to those states and so requires fewer steps, leading to a small STD.

We also verified the case of stochastic action selection following Eq. 3 for learning. The average number of steps required to solve a single trial using stochastic action selection without any prior learning is $3,072$. RL without any guidance or optimization would require many times this number of learning steps. In contrast, only 99 steps were performed by tele-operation, which would be required with real robot hardware. This advantage of several orders of magnitude enables real-world RL.

Table 1: Performance of two supervised RL methods after 300 off-line training trials for a grid world of size 25×25. Average (Avg.) number (#) of steps, standard deviation (STD) and 95% confidence interval ($95\%CI$).

State activation	Avg. # steps	STD	95%CI
Single activation	86.74	107.59	21.09
Gaussian activation	**36.01**	38.53	7.55
Tele-operated	33.00	6.93	7.84

5. Real-world docking scenarios and experimental results

After proving our techniques in simulation, we applied them to the two real-world docking scenarios described in Section 2. Results of both cases are presented below:

5.1. Backward docking station for autonomous recharging

Once the robot is 40 cm away from the docking position (see Fig. 2(b)), the two small landmarks placed on the docking station can reliably be detected and used for precise docking by the RL algorithm.

The state space is formed by the combination of three variables. These are the angular sizes of the two small naomarks and the yaw (pan) head angle. They encode the robot's distance and orientation relative to the docking station, respectively. The minimal allowed distance of the robot's camera to the landmark is approx. 13 cm, which corresponds to the robot's shoulder size plus a safety distance.

Those three values are discretized as follows. The angular size of each landmark within the visual field is discretized into 10 values for each landmark. These values represent distances in an interval of $[13\text{ cm}, 40\text{ cm}]$ in increments of 2.7 cm. We add one value for each landmark to indicate the absence of the corresponding landmark. This leads to a total of 11 values per landmark. The third variable is the head's pan angle. An internal routine permanently turns the robot's head to keep the interesting landmark centered in the visual field. The head movements are limited to $[70°, 120°[$ and the values are discretized with increments of 3.3° yielding 15 values. Hence, the total number of used states is obtained by the combination of all the values, i.e. $11 \times 11 \times 15 = 1815$.

The actions that the robot can perform are as follows: move forward and backward 2.5 cm, turn left or right 9° and move sideward to the left

Table 2: Summary of ten backward docking trials.

State activation	# of success	# of false positive	# of aborted	Avg. # of steps on success	STD
Single activation	6	1	3	23.80	8.23
Gaussian activation	8	1	1	**19.30**	8.35

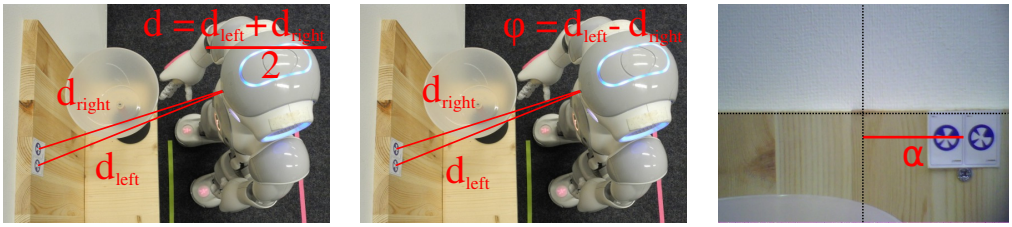
or right 2.5 cm. The turn and sideward movements are unfortunately very unreliable, which will be discussed later in Section 6. These values were adjusted empirically as a trade-off between speed and accuracy.

We tele-operate the robot from several random positions to the goal position saving the action state vectors and reward value. This training set with near-optimal routes is used for off-line learning. Specifically a total of 50 training examples with an average of 20 action steps were recorded. Then, using this training set, 300 trials were performed off-line, i.e. each of the 50 examples were presented 6 times. Table 2 summarizes the obtained results. We considered as *success* when the robot successfully reaches the desired goal position; as *false positive* when the robot perceives to be in the goal position but fails to make electrical contact with the charging station; and as *aborted* trial when the robot leaves away from the working space or collides with the docking station. The Gaussian activation led to more successful trials and a slightly reduced number of steps required during these trials.

5.2. Forward docking station for grasping

Similarly to the backward docking scenario, the state space is formed by the combination of three variables. In this case the variables are as follows: the average distance d to the two small naomarks measured in cm (estimated from the perceived size of the landmarks), the difference φ of the perceived distance between both naomarks, and the horizontal position α measured in radians between the center of the visual field and the naomarks array; see Fig. 6. They encode the robot's relative distance and orientation, respectively. Another difference to the backward docking is that the head remains fixed.

The three variables are discretized as follows: d is discretized into 14 values, representing distances within the interval [15 cm, 45 cm]. φ is discretized into 14 values. α ranges from $[-0.35, 0.35]$ in radians and is reduced to 10 values. Hence, the total number of states is obtained by the combination of



(a) (b) (c)

Figure 6: State space definition for the forward docking scenario.

Table 3: Summary of 25 forward docking trials.

State activation	# of success	# of aborted	Avg. # of steps on success	STD
Single activation	13	12	71.08	59.08
Gaussian activation	23	2	**13.70**	11.19

all the values, i.e. $14 \times 14 \times 10 = 1960$ states. We use the same actions as in the backward case.

We tele-operate the robot from 9 random positions to the goal position saving the action state vectors and reward value. The average number of steps required for tele-operated trials was 12. This training set with near-optimal routes is used for 300 off-line repetitions. We compare results obtained after 300 off-line learning trials with supervised single-state and supervised Gaussian activation. After the training phase using single state activation, the robot is able to reach the goal imitating the tele-operated routes, while the robot's actions remain random in those states that have not been visited. In contrast, after training with a Gaussian distributed state activation the robot is able to dock successfully from almost every starting point. Table 3 summarizes the obtained results.

Samples of obtained receptive fields (RFs) are presented in Fig. 7. The goal position is shown centered in the left side of each picture. Color intensity indicates weight strength, blue excitatory weights and red inhibitory weights. White pixels represent unlearned state-action pairs, which make the majority after training with single state activation. More intense colored pixels represent a stronger state-action binding and thus the action is more likely to be selected when the robot is in this state. When using Gaussian activation all weights have a non-zero value, although it may be small.

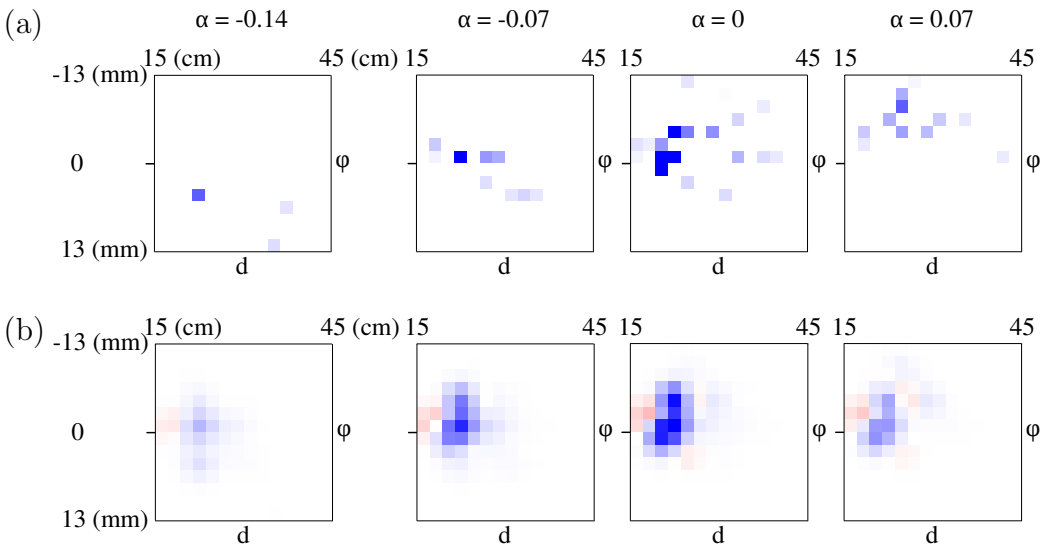


Figure 7: Receptive field (RF) samples of one action unit (*Move forward*) after learning. Color intensity represents the weight strength. Blue colors represent excitatory weights and red colors inhibitory weights. From left to right the RF samples for $\alpha \in \{-0.14, -0.07, 0, 0.07\}$ are presented. (a) Single state activation. (b) Gaussian state activation, $\sigma = 0.85$.

6. Discussion

We notice significant performance differences in the tested real-world scenarios. Specifically, for the autonomous recharging case, side movements are used as main actions, unfortunately, these movements were very unreliable, leading often to stand-still, slight turns or even side movements to the opposite direction. Furthermore, for backward docking, we used an automatic head repositioning to keep the landmarks centered in the visual field, and we used the robots' yaw angle as one of the variables to encode in the state space, which includes motor errors. Therefore the encoding of the state space is less precise than when keeping the head fixed and using the horizontal position of the landmark within the visual field, as done in the case of forward docking for grasping.

These two factors, inaccurate sideward movements and less precise state space definition, contributed to a lower success rate of the autonomous recharging behavior. This is why 50 tele-operated training examples were required to achieve acceptable results; see Table 2. However, the higher number of tele-operated examples implies that a larger portion of the state space has been covered, approx. 5%. This was not necessary in the case of forward docking and acceptable results were obtained using only 9 tele-operated examples, equivalent to approx. 1% coverage of the state space. This, of course, has an impact in the overall performance of supervised SARSA, but not much in supervised SARSA with Gaussian state activation. Note that we did not use any state space reduction technique.

Figure 8 presents a common problem to both scenarios, i.e. the effect of noisy sensory input and action execution. It shows 7 actions of a successful forward docking trial after off-line training using Gaussian state activation. The blue curve represents a reconstruction of NAO's perception of its position and orientation and the red curve shows the real NAO's position and orientation, as obtained from a ceiling camera. The letters inside the head-like shape denote the selected action. Of special interest are the cases of wrong perception. For example, when the NAO performs a right step from location 1 to location 2, he perceives a backward-directed movement, or, when turning left at location 6, he perceives a larger translation.

Tele-operation creates a few representative training examples to cover substantial parts of the state space and to speed up initial learning. Poor sampling from the state space during training would lead to poor initial performance in unexplored regions. A representative training set should consist

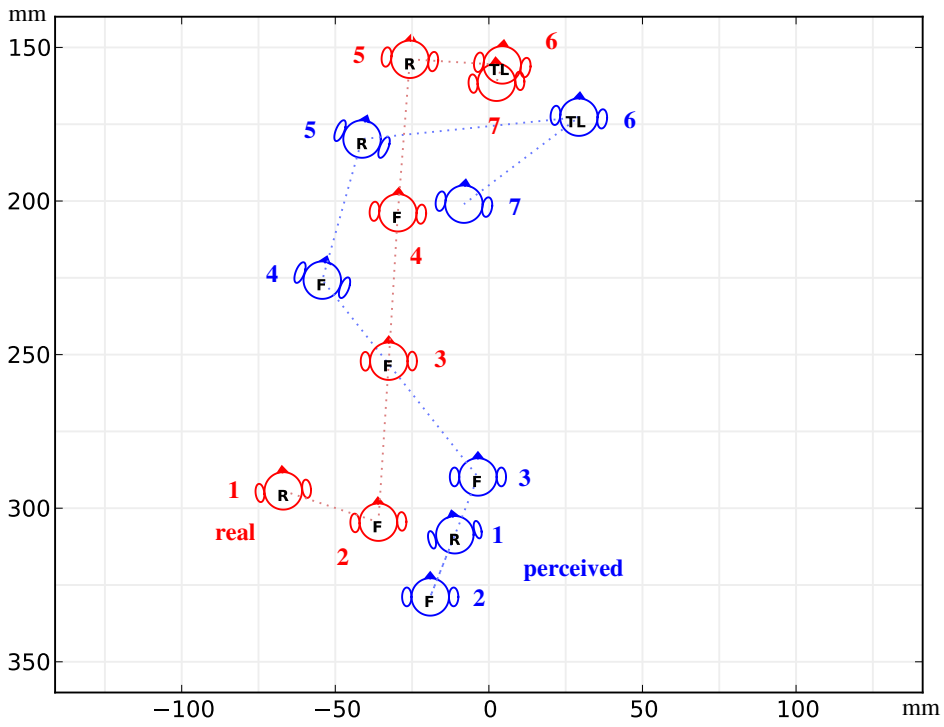


Figure 8: NAO's trajectory during forward docking after being trained using Gaussian state activation. The head-like shape represents NAO's position and orientation. Letters F, B, R, L, TR and TL denote forward, backward, move right, move left, turn right and turn left respectively, starting at the shown position. Blue represents NAO's perceived positions. Red represents NAO's real positions captured by a ceiling camera.

not only of the most frequent trajectories but it should particularly cover less frequently visited regions of the state space. A practical way to build a representative training set is in an incremental fashion, i.e. generate a training set, train the network and test the output placing the robot in a random position within the workspace. If the result is unsatisfactory then generate additional training examples by tele-operation containing the difficult case and re-train the network. These steps should be repeated until the results are satisfactory.

7. Conclusion

Motivated by the need for a precise docking behavior for the NAO robot and the suitability of RL techniques for navigation, we developed a real-world learning algorithm based on SARSA and supervised RL. We achieved a considerable reduction in the required learning steps from several thousand to a few hundred. The use of appropriate training examples proved to be a key factor for real-world learning scenarios, i.e. a representative sampling from the state space during tele-operation will contribute to the performance of the running system.

Additionally, Gaussian distributed state activation demonstrated to be useful for generalization and eliciting a state space reduction effect while not losing performance when applied to large state spaces. This technique reduces failures that may be induced by ambiguous or poor sampled state spaces. Furthermore, the use of a memory of successful action sequences may be of considerable value in other applications. This memory could be generated independently by tele-operation or fully automated operation. Then these examples can be used for automatic off-line training, while the robot is executing less demanding tasks.

Other well established methods for speeding up learning exist. For instance, TD(λ) accelerates learning by maintaining an eligibility trace of recently used states (in actor-critic learning), or state-action pairs (in SARSA), controlled by a trace decay parameter $0 \leq \lambda \leq 1$ [1]. Thereby, when δ becomes large at time t, not only the current, but also more recently visited state-action pairs, prior to t, of the current trial will be affected by the update. The number of trials required for learning can thereby decrease by an order of magnitude. In our model, however, we distinguish real-world trials of the robot from the repetition of stored sequences in an off-line mode. The repetitions have an effect similar to TD(λ) reducing the number of necessary

real-world trials (which is the important quantity in terms of costs). Moreover, while an eligibility trace only affects the most recent trial, repetitions affect all trials stored in memory, so dynamic programming can be performed on all stored real-world trials until convergence. Finally, the use of Gaussian activated states affects not only visited states but also neighboring states.

The proposed method was tested in two real-world scenarios; a partially backward docking used for autonomous recharging, which the robot can perform successfully, and a forward docking for a grasping task, which is under development. During the experimental phase, we noticed that 2-dimensional landmarks can be detected only from within a small angle range, i.e. when the robot sees them without much distortion, and detection is very noise susceptible. For future work a docking procedure using a 3-dimensional landmark is under development [18]. Additionally, forward, backward and turn movements have to be preferred, because of the limited effectiveness of sideward movements due to slippage of the NAO. The promising results shown in this paper motivate us to further improve the presented real-world scenarios and explore new applications.

Acknowledgments

This research has been partly supported by the EU projects RobotDoC [19] under 235065 ROBOT-DOC from the 7th Framework Programme (FP7), Marie Curie Action ITN, and KSERA funded from FP7 for Research and Technological Development under grant agreement n° 2010-248085. The authors thank Stefan Heinrich and Tayfun Alpay for proof reading and for contributing to testing in the real world, respectively.

References

[1] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction (adaptive computation and machine learning), The MIT Press, Cambridge, 1998.

[2] C. Weber, M. Elshaw, S. Wermter, J. Triesch, C. Willmot, Reinforcement learning embedded in brains and robots, in: Reinforcement learning: Theory and applications, InTech Education and Publishing, 2008, pp. 119–142.

[3] W. Schultz, P. Dayan, P. R. Montague, A neural substrate of prediction and reward, Science 275 (5306) (1997) 1593–1599.

[4] I. Ghory, Reinforcement learning in board games, Tech. rep., Department of Computer Science, University of Bristol (2004).

[5] J. Provost, B. J. Kuipers, R. Miikkulainen, Self-organizing perceptual and temporal abstraction for robot reinforcement learning, in: AAAI Workshop on Learning and Planning in Markov Processes, 2004.

[6] P. Zang, R. Tian, A. L. Thomaz, C. L. Isbell, Batch versus interactive learning by demonstration, in: Proceedings of the International Conference on Development and Learning (ICDL), IEEE, 2010, pp. 219–224.

[7] K. Conn, R. A. Peters, Reinforcement learning with a supervisor for a mobile robot in a real-world environment, in: International Symposium on Computational Intelligence in Robotics and Automation (CIRA), IEEE, Los Alamitos, 2007, pp. 73–78.

[8] K. Ito, Y. Fukumori, A. Takayama, Autonomous control of real snake-like robot using reinforcement learning; abstraction of state-action space using properties of real world, in: M. Palaniswami, S. Marusic, Y. W. Law (Eds.), Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information (ISSNIP), IEEE, Los Alamitos, 2007, pp. 389–394.

[9] T. C. Kietzmann, M. Riedmiller, The neuro slot car racer: Reinforcement learning in a real world setting, in: M. A. Wani, M. Kantardzic, V. Palade, L. Kurgan, Y. Qi (Eds.), International Conference on Machine Learning and Applications (ICMLA), IEEE, Los Alamitos, 2009, pp. 311–316.

[10] C. Weber, J. Triesch, Goal-directed feature learning, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), IEEE Press, Piscataway, NJ, USA, 2009, pp. 3355–3362.

[11] D. Muse, S. Wermter, Actor-critic learning for platform-independent robot navigation, Cognitive Computation 1 (3) (2009) 203–220.

[12] Nao academics edition: medium-sized humanoid robot developed by Aldebaran Robotics, http://www.aldebaran-robotics.com/.

[13] A. Louloudi, A. Mosallam, N. Marturi, P. Janse, V. Hernandez, Integration of the humanoid robot Nao inside a smart home: A case study, in: Proceedings of the Swedish AI Society Workshop (SAIS), Vol. 48 of Linköping Electronic Conference Proceedings, Uppsala University, Linköping University Electronic Press, 2010, pp. 35–44.

[14] The KSERA project (Knowledgeable SErvice Robots for Aging), http://ksera.ieis.tue.nl/.

[15] N. Navarro, C. Weber, S. Wermter, Real-world reinforcement learning for autonomous humanoid robot charging in a home environment, in: R. Groß, L. Alboul, C. Melhuish, M. Witkowski, T. Prescott, J. Penders (Eds.), Proceedings of the Annual Conference Towards Autonomous Robotic Systems (TAROS), Vol. 6856 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 231–240.

[16] W. Yan, C. Weber, S. Wermter, A neural approach for robot navigation based on cognitive map learning, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), 2012.

[17] D. Foster, R. Morris, P. Dayan, A model of hippocampally dependent navigation, using the temporal difference learning rule, Hippocampus 10 (1) (2000) 1–16.

[18] J. Kleesiek, A. K. Engel, C. Weber, S. Wermter, Reward-driven learning of sensorimotor laws and visual features, in: Proceedings of the IEEE International Conference on Development and Learning (ICDL), Vol. 2, IEEE, 2011, pp. 1–6.

[19] The RobotDoC collegium: The Marie Curie doctoral training network in developmental robotics, http://robotdoc.org/.