Real-World Reinforcement Learning for Autonomous Humanoid Robot Charging in a Home Environment

Nicolás Navarro, Cornelius Weber, and Stefan Wermter

University of Hamburg, Department of Computer Science, Knowledge Technology
Vogt-Kölln-Straße 30, D - 22527 Hamburg, Germany
{navarro,weber,wermter}@informatik.uni-hamburg.de
http://www.informatik.uni-hamburg.de/WTM/

Abstract. In this paper we investigate and develop a real-world reinforcement learning approach to autonomously recharge a humanoid Nao robot [1]. Using a supervised reinforcement learning approach, combined with a Gaussian distributed states activation, we are able to teach the robot to navigate towards a docking station, and thus extend the duration of autonomy of the Nao by recharging. The control concept is based on visual information provided by naomarks and six basic actions. It was developed and tested using a real Nao robot within a home environment scenario. No simulation was involved. This approach promises to be a robust way of implementing real-world reinforcement learning, has only few model assumptions and offers faster learning than conventional Q-learning or SARSA.

Keywords: Reinforcement Learning, SARSA, Humanoid Robots, Nao, Autonomous Docking, Real World.

1 Introduction

Reinforcement learning (RL) is a biologically supported learning paradigm [13, 14, 3], which allows an agent to learn through experience acquired by interaction with its environment. Reinforcement learning neural network architectures have an input layer, which represents the agent's current state, and an output layer, which represents the chosen action given a certain input.

Reinforcement learning algorithms usually begin with the agent's random initialization followed by many randomly executed actions until the agent eventually reaches the goal. Following a few successful trials the agent starts to learn action-state pairs based on its acquired knowledge. The 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 does not use feedback for intermediate steps, but rather a reward (or punishment) is 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). The limited feedback characteristics of this learning approach make it a relatively slow learning mechanism, but attractive due to its potential to learn action sequences.

In the literature, reinforcement learning is usually used within simulated environments or abstract problems [15, 5, 11]. Those kinds of problems require a model of the agent-environment dynamics, which it is not always available or easy to infer. Moreover, a number of assumptions, which are not always realistic, have to be made, e.g. action-state transition model, design of reward criterion, magnitude and kind of noise if any, etc.

On the other hand, real-world reinforcement learning approaches are scarce [2, 6, 7], mostly, because RL is expensive in data or learning steps and the state space tends to be large. Moreover, real-world problems present additional challenges, such as safety considerations, real time action execution, changing sensors, actuators and environmental conditions, among many others.

Several techniques to improve real-world learning capabilities of RL algorithms exist. *Dense reward functions* [2] provide performance information in intermediate steps to the agent. Another frequently used technique is manual *state space reduction* [2, 7], which is a very time consuming task. Other approaches propose modification and exploitation of the agent's properties [6], which is not always possible. *Batch reinforcement learning* algorithms [7] use information from past state transitions, instead of only the last transition, to calculate the prediction error function; this is a powerful approach but a computationally demanding technique. A final example of these techniques is supervised reinforcement learning algorithms [2]. The supervision consists of human-guided action sequences during initial learning stages.

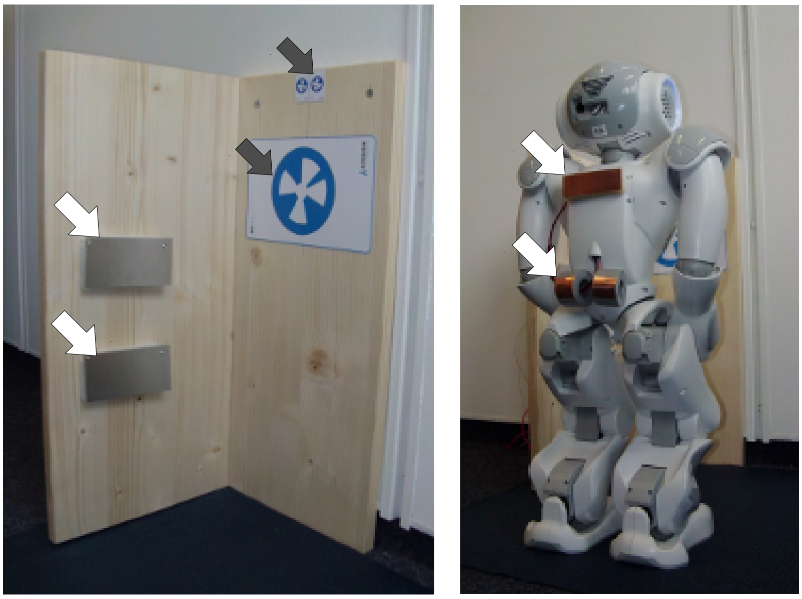
The proven value of RL techniques for navigation and localization tasks [10, 2] motivates us to develop a RL approach to navigate autonomously into a docking station used for recharging. This approach makes use of a supervised RL algorithm and a Gaussian distributed state activation that allows real-world RL. Our approach proves to work with a reduced number of training examples, and is robust and easy to incorporate into conventional RL techniques such as SARSA.

2 Problem Overview

There are a number of research approaches studying domestic applications of humanoid robots, in particular using the Nao robot [9, 8]. One of the Nao's limitations for this kind of environment is due to its energetic autonomy, which typically does not surpass 45 min. This motivates the development of strategies to increase the robot's operational time minimizing human intervention. In this work we develop a real-world reinforcement learning based on SARSA learning, see section 3, applied to an autonomous recharging behavior. This work is validated using a real Nao robot inside a home-like environment.

Several docking station designs and/or recharging poses are possible. The proposed solution is intended to increase the energetic capabilities of the Nao without major interventions on the robot's hardware or affecting its mobility or sensory capabilities. Despite the challenge to maneuver the robot backwards, we chose a partial backward docking. This offers advantages such as easy mounting on the Nao, it does not limit the robot mobility, nor obstructs any sensor, nor requires long cables going to the robot extremities and allows a quick deployment after the recharging has finished or if the robot is asked to do some urgent task.

The prototype built to develop the proposed autonomous recharging is shown in figure 1(a). White arrows indicate two metallic contacts for the recharging, and gray arrows indicate three landmarks (naomarks)[1] used for navigation. The big landmark is used when the robot is more than 40 cm away from the charging station, while the two smaller landmarks are used for an accurate docking behavior.



(a) Charging station

(b) Nao robot with electrical contacts

Fig. 1. (a) White big arrows indicate the electrical contacts placed on the docking station and gray arrows indicate the landmarks position. (b) Robot's electrical connections.

The autonomous recharging was split into four phases. During the first phase a search and approach hard-coded algorithm searches for the charging station via a head scan followed by a robot rotation. The robot estimates the charging station's relative position based on geometrical properties of landmarks and

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

moves towards the charging station. This approach places the robot approximately 40 cm away from the landmarks, see figure 2(a). In the second phase the robot re-estimates its position and places itself approximately parallel to the wall as shown in figure 2(b).

The third phase uses the reinforcement learning (SARSA) algorithm to navigate the robot backwards very close to the electric contacts as presented in figure 2(c).[2] After reaching the final rewarded position, the fourth and final phase starts. A hard-coded algorithm moves the robot to a crouch pose, see figure 2(d), in which the motors are deactivated and the recharging starts.

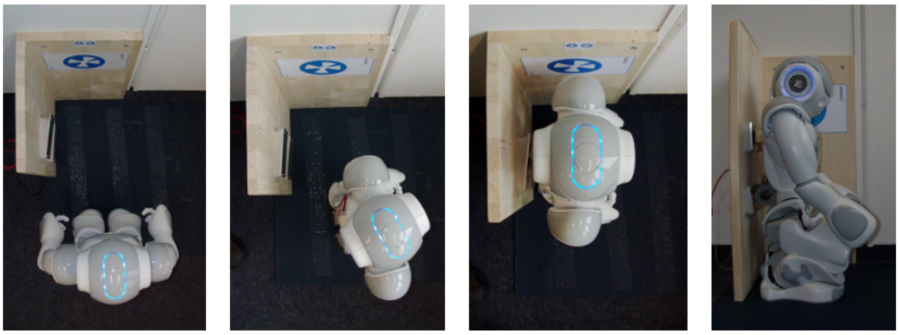

(a) approach (b) alignment (c) docking (d) crouch pose

Fig. 2. Top view of the autonomous robot behavior in its four different phases (approaching, alignment, docking and recharging).

3 Network Architecture and Learning

We use a fully connected two layer neural network, see figure 3. The input layer (1815 neurons) represents the robot's relative distance and orientation to the landmarks. The output layer (6 neurons) represents the actions that can be performed: move forward and move backward 2.5 cm, turn left or right 9° and move sideward to the left or right 2.5 cm. These values were adjusted empirically as a trade-off between speed and accuracy.

As mentioned in section 2, the robot starts to execute the SARSA algorithm approx. parallel to the wall and 40 cm away from the landmark.[3] During docking the minimal measured distance of the robot's camera to the landmark is approx. 13 cm, which corresponds to the robot's shoulder size plus a small safety distance. 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 represent the robot's relative distance and orientation, respectively.

[2] In this docking phase, Nao's gaze direction is oriented towards the landmarks
[3] Distance measured from the landmark to the robot's camera

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 from [13, 40] cm in intervals of 2.7 cm. We add 2 values 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 intervals of 3.3° yielding 15 new values. Hence, the total number of used states is obtained by the combination of all the values, i.e. $11 * 11 * 15 = 1815$.

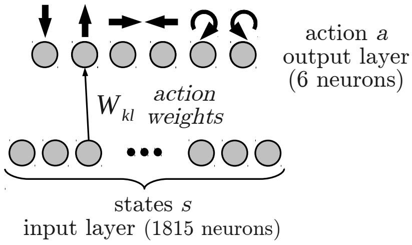


Fig. 3. Neural network schematic overview. An example of connections in the used neural network.

The learning algorithm is based on SARSA [13, 14] and summarized as follows.

For each trial the robot is placed at an initial random position within the detection area. It permanently turns its head towards the landmarks. The landmarks sizes and the head pan angle are used to compute the robot internal state. Here, instead of using a single state activation of SARSA, where only a single input neuron has maximal activation ($S_i = 1$) at the time, we use a Gaussian distributed activation of states [4], which is centered in the current robot internal state ("*SARSA active state*"). The Gaussian is normalized, i.e. the sum over the state space activations is 1.

$$S_j = \frac{1}{\sigma^3 (2\pi)^{2/3}} \cdot e^{-\dfrac{(x_j - \mu_x)^2 + (y_j - \mu_y)^2 + (z_j - \mu_z)^2}{2\sigma^2}} \tag{1}$$

We use $\sigma = 0.85$, which effectively "blurs" the activation around the "*SARSA active state*". In this way generalization to states that have not been visited directly is possible.

μ_x represents the current size value for *"landmark 1"*, μ_y represents the current size value for *"landmark 2"* and μ_z represents the current value for the head yaw angle. The variables x_j, y_j and z_j take all the possible values of the respective dimension, i.e. size "*landmark 1*", size "*landmark 2*" and head's pan angle, respectively. In this way a normalized state activation is computed centered around (μ_x, μ_y, μ_z) and extend to the entire state space.

One of the motivations of having a Gaussian state activation is that states closer to the current internal state are more likely to generate the same action than farther states. Using this idea, we can extend and spread what we know about a state over larger regions of the state space.

Poor sampling from the state space during training will lead to poor generalization. This can be compensated using a representative training example set generated by tele-operation, what is termed supervised reinforcement learning [2]. A representative training example set should consist not only of the most frequent trajectories but it should cover also less frequently visited regions of the state space. A practical way to build a representative training example 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 (ideally not contained in the training examples). If the result is unsatisfactory then generate a few additional training examples containing this troubled case and re-train the network. These steps should be repeated until the results are satisfactory.

With the input from equation (1), the net activation of action unit i is computed as:

$$h_i = \sum_l W_{il} S_l \tag{2}$$

W_{ij} is the connection weight between action i and state l. Connection weights are initially set to zero. Next, we used a softmax-based stochastic action selection:

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

β controls how deterministic the action selection is, in other words the degree of exploration of new solutions. Large β implies a more deterministic action selection or a greedy policy. Small β encourages the exploration of new solutions. We use $\beta = 70$ to prefer new routes. Based on the activation state vector (S_l) and on the current selected action (a_k), the value $Q_{(s,a)}$ is computed:

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

A binary reward value r is used. If the robot reaches the desired position it is given $r = 1$, zero if it does not. The prediction error based on the current and previous $Q_{(s,a)}$ value is given by:

$$\delta = (1 - r)\gamma Q_{(s',a')} + r - Q_{(s,a)} \tag{5}$$

The time-discount factor γ controls the importance of proximal rewards against distal rewards. Small values are used to prioritize proximal rewards. On the contrary, values close to one are used to consider equally all rewards. We use $\gamma = 0.65$. The weights are updated using a δ-modulated Hebbian rule with learning rate $\epsilon = 0.5$:

$$\Delta W_{ij} = \epsilon \delta a_i S_j \tag{6}$$

4 Supervised Reinforcement Learning and Experimental Results

In real-world scenarios, the random exploration of the state space, common in reinforcement learning, is prohibitive for several reasons such as real time action execution, safety conditions, changing sensors, actuators and environmental conditions, among many others.

In order to make the docking task feasible in a real-world RL approach, we skip the initial trial and error learning as presented in [2]. 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 non-optimal routes is used for *offline* learning. Specifically 50 training examples with an average of 20 action steps were recorded. Then, using this training set, 300 trials were computed. Within each trial, the SARSA learning algorithm was performed as described in equations (1)-(6), however in equation (3) the selected action was given by the tele-operation data. We refer to this procedure as supervised RL. The state activation was tested for three cases: using conventional single state activation, using Gaussian distributed state activation and using a truncated Gaussian state activation. The truncated Gaussian state activation is obtained by limiting the non-zero values of x, y and z to a neighborhood of 1 state radius around μ_x, μ_y, and μ_z respectively and then normalized, in other words apply the Gaussian distribution to a neighborhood of one state radius around the "*SARSA active state*" instead of applying the activation over the entire state space.

We compare results obtained with the weight for each case after 300 trials. After the training phase using single states activation, the robot is able to reach the goal imitating the tele-operated routes. However, the robot's actions turn random in the states that have not yet being visited. In contrast, after training with a Gaussian distributed state activation the robot is able to dock successfully from almost every starting point, even in those cases where the landmarks are not detected in one step. This provides the Gaussian state activation with a clear advantage in terms of generalization. Thus faster learning than in case of SARSA or Q-learning is obtained. For the truncated Gaussian activation we observe slightly better results than using single state activation. A partial Gaussian activation may be useful for instance when the states are very different to each other and thus different actions are required.

In table 1, we compare the performance of the three methods starting from ten different positions. We present the number the steps executed in each trial and the corresponding stopping condition. *Single* stands for SARSA state activation. *Truncated* stands for Gaussian state activation limited to a neighborhood of one state radius around SARSA active state. *Gaussian* stands for Gaussian distributed state activation. We consider as *Success*, when the robot reaches successfully the desired goal position; as *False Pos.* (false positive) when the robot's measurement indicates that it is in the goal position but is not touching the metallic contacts. *Blind* indicates when the robot hadn't seen the landmarks for three or more consecutive actions. *Collision* indicates that the robot was crashing against the docking station. Under a detected *Blind* or *collision* event the respective trial was aborted.

Table 1. Results of the three tested methods for ten different trajectories.

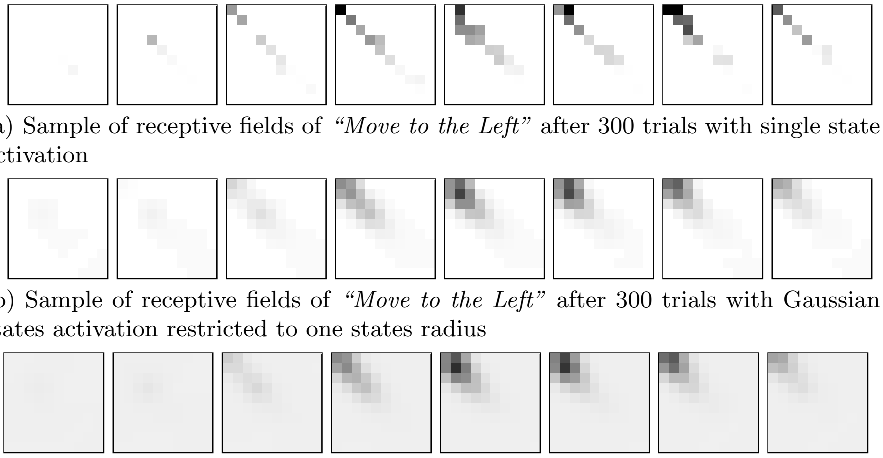
	Single		Truncated		Gaussian	
Starting position	Stop condition	Step Nr.	Stop condition	Step Nr.	Stop condition	Step Nr.
1	Success	15	False Pos.	7	Success	9
2	False Pos.	5	Success	9	Success	17
3	Success	28	Success	29	Success	10
4	Success	20	Success	33	Success	13
5	Blind	17	Blind	7	Success	22
6	Blind	17	Blind	17	Success	32
7	Success	32	Success	48	Collision	49
8	Success	33	Success	23	False Pos.	37
9	Collision	154	Success	9	Success	24
10	Success	15	Success	14	Success	27

Table 2 summarizes the obtained results. We present the average number of steps needed to reach the goal after training. The learned action-state pairs indicate the percentage of network weights that differ from their initialization values.

Table 2. Summary of ten trials for the three tested methods.

State activation	Action-state pairs learned (%)	Nr. of success	Nr. false positive	Nr. aborted	Avg. nr. steps on success	Std. deviation
Single	4	6	1	3	23.8	8.23
Truncated	34	5	3	2	23.6	14.3
Gaussian	100	8	1	1	19.3	8.35

Examples of the obtained receptive fields (RFs) after 300 trials are presented in figure 4. The goal position is shown in the upper left corner of each picture. White pixels represent unlearned action-state pairs. Darker gray represent a stronger action-state binding and thus the action is more likely to be selected when the robot is in this state. The eight different pictures for each case correspond to the different action-state pairs for particular head angles.



(a) Sample of receptive fields of *"Move to the Left"* after 300 trials with single state activation

(b) Sample of receptive fields of *"Move to the Left"* after 300 trials with Gaussian states activation restricted to one states radius

(c) Sample of receptive fields of *"Move to the Left"* after 300 trials with Gaussian states activation, $\sigma = 0.85$ without cutoff

Fig. 4. Receptive fields (RFs) of one action unit (*Move to the Left*) after 300 trials. Dark color represents the weight strength. From left to right the RFs for 8 of the 15 possible head rotations are presented.

5 Conclusions

Motivated by the limited energetic capabilities of the Nao robot and our need for studying humanoid robots within home environments, we developed an autonomous navigation procedure for recharging the Nao, which does not require human assistance. Autonomous docking for a Nao robot was achieved for a real home like environment. Initial training examples, together with a Gaussian distributed states activation made real-world learning successful.

The use of appropriate training examples proved to be a key factor for real-world learning scenarios, reducing considerably the required learning steps from several thousand to a few hundred. Additionally, Gaussian distributed states activation demonstrated to be useful for generalization and eliciting a state space reduction effect. The use of these techniques is straightforward to SARSA learning. Promising results were presented, which suggest further opportunities in real-world or simulated scenarios.

We see at least two possible extensions of Gaussian distributed states activation. We believe that a conservative version, extending only to a small neighborhood called *truncated* in section 4, could help to increase learning speed even without tele-operated examples. Alternatively, the use of a memory of successful action sequences may be of great utility in other applications. This memory could be generated independently by tele-operation or fully automatic. Then these examples can be used for automatic offline training, while the robot is executing less demanding tasks.

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. Additionally, forward, backward and turn movements will be preferred, because of the limited performance of sideward movements due to slippage of the Nao.

To obtain a more robust solution using this approach, we suggest adding a final module after the reinforcement learning module. The objective of this module will be to check sensor values, including sensors not considered in the current implemented, to determine whether the robot is in a false positive position. In this case, corrective actions could be learnt.

Acknowledgments This research has been partly supported by the EU project RobotDoc [12] under 235065 ROBOT-DOC from the 7th Framework Programme, Marie Curie Action ITN and by the KSERA project funded by the European Commission under the 7th Framework Programme (FP7) for Research and Technological Development under grant agreement n° 2010-248085.

References

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

[2] Conn, K., Peters, R.A.: 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). pp. 73–78. IEEE, Los Alamitos (2007)

[3] Dorigo, M., Colombetti, M.: Robot shaping: An experiment in behavior engineering (intelligent robotics and autonomous agents). The MIT Press, Cambridge (1997)

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

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

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

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

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

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

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

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

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

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

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

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