Q-RAN: A Constructive Reinforcement Learning Approach for Robot Behavior Learning

Li Jun, Achim Lilienthal
AASS, Department of Technology
Örebro University
SE-701 82 Örebro, Sweden
Email: li.jun@tech.oru.se
Email: achim@lilienthals.de

Tomás Martínez-Marín
Department of Physics, System
Engineering and Signal Theory
University of Alicante
Alicante, Spain
Email: tomas@dfists.ua.es

Tom Duckett
Department of Computing
and Informatics
University of Lincoln
Lincoln LN6 7TS, UK
Email: tduckett@lincoln.ac.uk

Abstract— **This paper presents a learning system that uses Q-learning with a resource allocating network (RAN) for behavior learning in mobile robotics. The RAN is used as a function approximator, and Q-learning is used to learn the control policy in 'off-policy' fashion that enables learning to be bootstrapped by a prior knowledge controller, thus speeding up the reinforcement learning. Our approach is verified on a PeopleBot robot executing a visual servoing based docking behavior in which the robot is required to reach a goal pose. Further experiments show that the RAN network can also be used for supervised learning prior to reinforcement learning in a layered architecture, thus further improving the performance of the docking behavior.**

I. Introduction

Using reinforcement learning (RL) for robot behavior learning is often confronted with two difficulties: representation of large continuous sensory spaces and the necessity of speeding up the learning process online in real time. A common practice to deal with the first problem is using a function approximator, such as artificial neural networks (ANNs) for approximating value functions, but usually requires much work on designing the network architecture and refining of the network parameters. The second problem usually involves prior knowledge to bias the learning process in order for the robot to learn a required behavior in a feasible manner and tolerable time.

This paper investigates a learning system that incorporates Q-learning [19] with a resource allocating network (RAN) [10] for robot behavior learning (thus, named as Q-RAN learning). Specifically, the RAN can automatically and dynamically grow its hidden neurons online to accommodate the training data from the robot (e.g., sonar, laser, and visual images) for the required behavior, thus simplifying the engineering process of the network structure and parameters. And the "off-policy" learning property of the Q-learning algorithm (meaning that the actions that the robot actually takes may be unrelated to the policy that is evaluated and improved [17]) enables learning to be bootstrapped by a "prior knowledge controller" [3], thus speeding up the reinforcement learning.

Our learning system is experimentally verified on a PeopleBot robot executing a visual servoing based docking behavior in which the robot is required to approach a table in order to grasp an object (see Fig. 2).

II. Related Work

There has been some research on using growing neural networks in reinforcement learning. A more recent work was done by Rivest and Precup [12], where TD-learning was combined with cascade-correlation networks to dynamically represent the state space based on the training data and tested on the Tic-Tac-Toe problem. Rivest and Precup systematically compared their growing network algorithm with some static neural networks such as *online backpropagation* and *batch-cached backpropagation*, and claimed that the combination of TD-learning with cascade-correlation networks performs better than static backpropagation networks. One problem is that the cache size of the look-up table in the cascade-correlation networks has to be decided *in advance* and can become intractable in a higher dimensional state space.

In the domain of robotics, Santos and Touzet used a growing RBF network to acquire a wall following behavior [13], where the network is used to directly calculate the actions as the output instead of approximating the action values (i.e., Q-values). Thus the application of Q-learning is not straightforward. Furthermore, the network structure could become complicated for a large state space because the number of the neurons in the output layer is the number of possible actions for the given problem.

Bruske et al. [2] built a sophisticated learning system based on the integration of a growing and pruning network called dynamic cell structures (DCS), a REINFORCE algorithm[22], and an adaptive heuristic critique (AHC) for learning an obstacle avoidance behavior. One potential problem of their learning system is that, as has been observed by Ratitch and Precup [11], pruning the network architecture does not work well with RL since the deleted neurons are usually the ones corresponding to the critical states (e.g., the goal state, or the wrong state) because they are visited *much less* during the learning process. In addition, the networks structure is complicated with many sub-architectures for adding the fuzzy rules as the priori knowledge, thus making it difficult to use. Note that all three reinforcement learning systems mentioned here based on growing networks do not support off-policy learning (see page 17–37 in [4]).

There has been much research on how to bias reinforcement learning with some prior knowledge in order to make the training time more realistic for robot leaning. For example, Smart and Kaelbling used the “programming by demonstration (PbD)” strategy in which the robot was directly driven to the interesting areas of the state space for a corridor following behavior by a human operator during the early stage of the learning process, thus generating a demonstration controller that initializes the Q-values for the subsequent reinforcement learning [16]. Our learning system differs from theirs in two ways: (1) their learning system involves two separated learning stages, offline demonstration controller learning and online reinforcement learning, whereas our learning system embeds the prior knowledge controller into the Q-learning, thus the reinforcement learning and prior knowledge controller are executed simultaneously; (2) their offline demonstration controller and corresponding state-action representation need be the same as that of the learning controller since they both are derived from the reinforcement learning framework, whereas our learning system can use any kind of prior knowledge controller.

Martinez and Duckett used a linear controller to speed up the training process for a visually-guided docking behavior on a PeopleBot robot [8]. In fact, using Q-RAN learning for visual servoing based docking in this paper is mainly motivated by their work, with the difference that their learning system is built on the discrete and predefined adjoining cell mapping structure (ACM) for Q-learning, while our Q-RAN learning system uses the continuous state space variables and can autonomously grow its own network structure for value function approximation.

We notice that using reinforcement learning for docking on the same platform has also been independently investigated by Weber et al. [20]. However in their learning system the camera is fixed so as to see both the gripper and the object to be grasped without visual servoing for camera control, thus the docking behavior is limited to a short distance of $0.4 - 0.5m$. By contrast, the docking behaviour acquired by the Q-RAN learning system with visual servoing is valid over a range of up to $3.0 - 4.0m$. In addition, the neural network used for image processing is also static in their learning system.

III. The Learning System

Our learning system is built on Watkins' one step Q-learning [17] and the RAN network [10] with the following characteristics.

First of all, by using “off-policy” learning, the Q-learning process in our learning system is speeded up by a prior knowledge controller. Here we assume that the prior knowledge controller can be easily acquired or quickly formulated for a “rough control” of the required behavior such as a direct input-output controller (e.g., a linear controller obtained from control theory for docking in [8], a previously learned neural network from supervised learning for goal finding [3]), a set of control rules formulated from fuzzy logic for wall following [9], or an initialization of Q-values acquired by the PbD approach [16]. Then, the reinforcement learning is applied to tune and refine the prior knowledge controller for the acquisition of the “optimal and robust” controller for the required behavior. In this paper we use a linear control model for the speed-up of Q-learning for the docking behavior, as done in [8] with some minor changes.

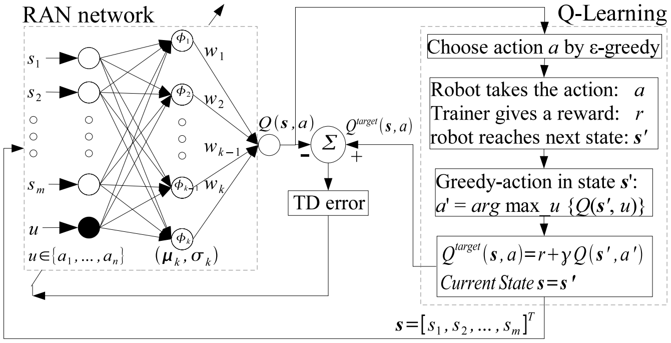

RAN network
s_1
s_2
s_m
u
$u \in \{a_1, \ldots, a_n\}$
ϕ_1
ϕ_2
ϕ_{k-1}
ϕ_k
$(\boldsymbol{\mu}_k, \sigma_k)$
w_1
w_2
w_{k-1}
w_k
$Q(\boldsymbol{s}, a)$
Σ
-
+
$Q^{target}(\boldsymbol{s}, a)$
TD error
Q-Learning
Choose action a by ε-greedy
Robot takes the action: a
Trainer gives a reward: r
robot reaches next state: $\boldsymbol{s}'$
Greedy-action in state $\boldsymbol{s}'$:
$a' = arg$ max_u $\{Q(\boldsymbol{s}', u)\}$
$Q^{target}(\boldsymbol{s}, a) = r + \gamma Q(\boldsymbol{s}', a')$
Current State $\boldsymbol{s} = \boldsymbol{s}'$
$\boldsymbol{s} = [s_1, s_2, \ldots, s_m]^T$

Fig. 1. Q-learning system with RAN network

The functionality of RAN as the function approximator is twofold in our Q-RAN learning system. Firstly, RAN provides a continuous representation for continuous state spaces, which enables the learning system to generalize the learned policy to unvisited states. More importantly, the nature of its automatic growing structure significantly reduces the engineering process of the state space representation. Unlike the three growing networks discussed in section I, our Q-RAN learning architecture is a simple three layer RBF network with only one output neuron for approximating the Q-value for a given state-action situation, thus making it easy to use.

Based on the above discussion and considerations, the Q-RAN learning system is depicted in Fig. 1, where the robot is in state $\boldsymbol{s} = [s_1, s_2, \ldots, s_m]^T \in R^m$, and its available actions are $u \in A = \{a_1, a_2, \ldots, a_n\}$ at time t. The computational steps of the learning algorithm are detailed as follows:

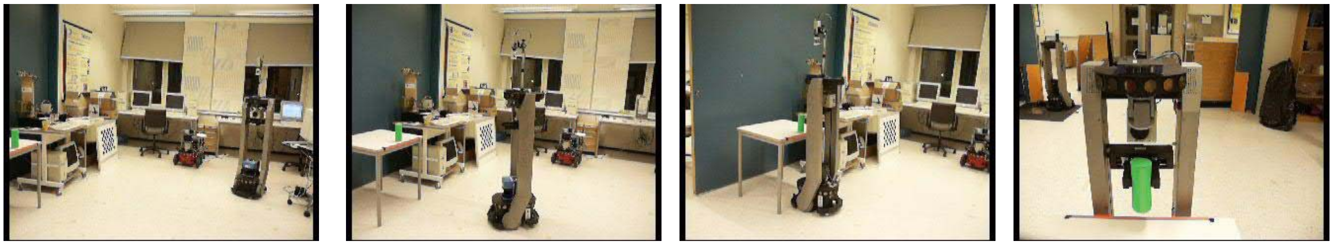
Initialize: learning rate η with initial value η_0, search factor τ, error threshold ϵ, novelty threshold d with initial value d_{max}, and $d \in [d_{min}, d_{max}]$, optimal receptive factor ρ and its overlap factor κ for the RAN network; discount factor γ, exploration rate ε with initial value ε_0, maximum steps in one trial t_{max}, and maximum number of trials T_{max} for Q-learning. Put robot in a starting position and get current state $\boldsymbol{s}$.

Note that at time step $t = 0$ in the first trial episode (i.e., $T = 0$), the action-values $Q(\boldsymbol{s}, u)$ are zeros for all actions $u \in \{a_1, a_2, \ldots, a_n\}$, and the RAN network starts with $K = 0$ radial basis functions defined as:

$$\begin{aligned}
&\boldsymbol{\phi}(\boldsymbol{s}, u) = [\phi_1(\boldsymbol{s}, u), \phi_2(\boldsymbol{s}, u), \ldots, \phi_K(\boldsymbol{s}, u)]^T, \\
&\phi_k(x) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_k\|^2}{\rho\sigma_k^2}\right), \\
&\boldsymbol{w} = [w_1, w_2, \ldots, w_K]^T,
\end{aligned} \tag{1}$$

where $\boldsymbol{\mu}_k = [\mu_{k1}, \mu_{k2}, \ldots, \mu_{k,m+1}]^T$ is the position vector, σ_k is the receptive field of the k-th radial basis function, and $\boldsymbol{x} = [\boldsymbol{s}, u]^T = [x_1, x_2, \ldots, x_{m+1}]^T$ is the input vector to the RAN network, which is constructed from $\boldsymbol{s}$ and u, as in Fig. 1.

1. **Calculate** the action-value $Q(\boldsymbol{s}, u)$ for all actions $u \in A$



(a) PeopleBot at start position (b) Tracking green can (c) Approaching table (d) Grasping can at goal posture

Fig. 2. Docking Behavior on the real robot.

using the RAN network, i.e.,

$$Q(\boldsymbol{s},u)=\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{s},u)+b,\ \text{for all } u\in A, \tag{2}$$

where b is the bias for RAN's output.

2. **Choose** an action using an ε-soft policy:

$$\begin{aligned}&a=\text{a random action}\in A \text{ with probability: } \varepsilon,\\ &a=\text{prior knowledge controller's output with probalility: } 1-\varepsilon.\end{aligned} \tag{3}$$

3. **Take** action a, observe reward r, and next state $\boldsymbol{s}'$.

4. **Calculate** the greedy action a' of the next state $\boldsymbol{s}'$ as

$$a'=\arg\max_{u\in A}(Q(\boldsymbol{s}',u), \tag{4}$$

where $Q(\boldsymbol{s}',u)=\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{s}',u)+b,$ for all $u\in A$.

5. **Calculate** the target action-value $Q^t(\boldsymbol{s},a)$ and the temporal difference(TD) error δ as,

$$Q^t(\boldsymbol{s},a)=r+\gamma Q(\boldsymbol{s}',a'), \quad \delta=Q^t(s,a)-Q(s,a). \tag{5}$$

6. **Construct** the radial basis functions for the input vector $\boldsymbol{x}$, If $K=0$, add the first neuron to RAN by setting:

$$\{\boldsymbol{\mu}_1=\boldsymbol{x},\sigma_1=\kappa d_{max},w_1=0,b=\delta,K=1\}. \tag{6}$$

Construct K radial basis functions:

$$\begin{aligned}\boldsymbol{\phi}(\boldsymbol{x})&=[\phi_1(\boldsymbol{x}),\phi_2(\boldsymbol{x}),\ldots,\phi_K(\boldsymbol{x})]^T,\\ \phi_k(\boldsymbol{x})&=\exp(-\|\boldsymbol{x}-\boldsymbol{\mu}_k\|^2/(\rho\sigma_k)^2).\end{aligned} \tag{7}$$

7. **Find** the distance ℓ between the input vector $\boldsymbol{x}'$ and the best matching neuron as

$$\ell=\min_{k=1\sim K}\|\boldsymbol{x}-\boldsymbol{\mu}_k\|. \tag{8}$$

8. **If** $\delta>\epsilon$ and $\ell>d$, insert a new neuron by setting:

$$\{\boldsymbol{\mu}_{new}=\boldsymbol{x},\sigma_{new}=\kappa\ell,w_{new}=\delta,K\Leftarrow K+1\}. \tag{9}$$

9. **Else** update the RAN's parameters as

$$\begin{aligned}&\boldsymbol{w}\Leftarrow\boldsymbol{w}+\eta\delta\boldsymbol{\phi}(\boldsymbol{x}), \quad b\Leftarrow b+\eta\delta,\\ &\mu_{ki}\Leftarrow\mu_{ki}+\eta\delta\phi_k(\boldsymbol{x})w_k\frac{x_i-\mu_{ki}}{\sigma_k}.\end{aligned} \tag{10}$$

10. **Decrease** the novelty threshold d and learning rate η of the RAN network, the exploration rate ε and learning rate α of the Q-learning as

$$\begin{aligned}&d\Leftarrow de^{-1/\tau}, \quad \text{if } d>d_{min},\\ &\eta\Leftarrow\eta_0e^{-1/\tau}, \quad \varepsilon\Leftarrow\varepsilon_0e^{-1/\tau}.\end{aligned} \tag{11}$$

11. **Set** the current state s, time t, and trial counter T as

$$\boldsymbol{s}=\boldsymbol{s}',\ t=t+1,\{T=T+1 \text{ if } t=t_{max}\}. \tag{12}$$

12. **Goto** step 1 until some stopping criteria are reached (e.g., the required behavior is achieved, or $T>T_{max}$).

One comment regarding the Q-RAN learning system is that the calculation of many Gaussian basis functions in formula 1 is an extremely time-consuming business when the number of radial basis functions K is large. Instead of a Gaussian function, a simple quadratic function is used for speeding up the computation, as done in [10], [5]:

$$\phi_k(\boldsymbol{x})=\begin{cases}(1-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}_k\|}{\rho\sigma_k^2})^2, & \text{if } \|\boldsymbol{x}-\boldsymbol{\mu}_k\|<\rho\sigma_k^2,\\ 0 & \text{otherwise.}\end{cases} \tag{13}$$

IV. Experiments on the Robot

A. Behavior Investigated

We investigated a docking behavior in which an ActivMedia PeopleBot robot is required to approach a table at a perpendicular angle for its gripper to grasp an object, using its pan-tilt camera as the only sensor. This behavior belongs to the class of episodic tasks, meaning that the robot starts in an arbitrary state, and eventually ends in the terminal state (goal state or failure state) [17]. Fig. 2 shows a successful episode of the docking behavior while the robot is under the control of our Q-RAN learning system. Fig. 2(a) shows the robot at the starting posture. The maximum range in which the docking behavior can be applied is 3 – $4m$ limited by the image resolution of 320×240 pixels. Fig. 2(b) shows the robot tracking the can and servoing its pan-tilt camera to keep the can in the center of the visual image. The pan and tilt ranges of the camera are set to $[-90°,90°]$ and $[20°,80°]$ with respect to the robot's orientation. Fig. 2(c) shows the robot reaching the goal pose with its camera being fully tilted down so as to face the can. Fig. 2(d) shows the robot grasping the can. The grasping action is hand-coded in the experiments.

The geometry of the problem is shown in the left column of Fig. 3. The robot and its goal pose are predefined in the global coordinates frame as $\{x_G,y_G\}$. Let $\alpha\in[-90,90]$ denote the angle between the robot heading direction x_R and the straight line P connecting the robot current position to the goal position, $\beta\in[-90,90]$ the angle between x_G axis and the line segment that is perpendicular to the line P, and $\{v_{trans},v_{rot}\}$ is the robot's translational velocity and rotational velocity.

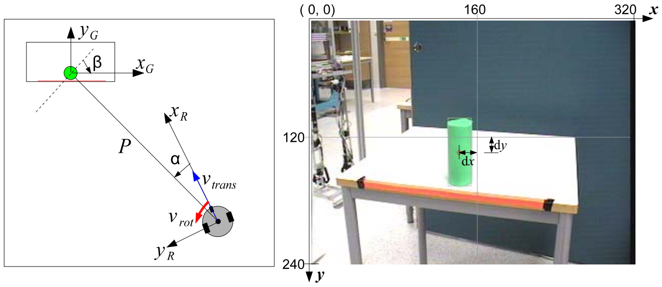


Fig. 3. The robot kinematics and the camera visual servoing

Based on the following kinematic equation,

$$\begin{bmatrix} \dot{P} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 \\ \sin\alpha/P & -1 \\ -\sin\alpha/P & 0 \end{bmatrix} \begin{bmatrix} v_{trans} \\ v_{rot} \end{bmatrix}, \qquad (14)$$

a simple linear controller,

$$\begin{aligned} v_{trans} &= k_P P, \\ v_{rot} &= k_\alpha \alpha + k_\beta \beta, \end{aligned} \qquad (15)$$

can drive the robot to the goal pose provided that the gain conditions $\{k_P > 0,\ k_\alpha - k_P > 0,\ -k_\beta > 0\}$ hold [15].

However, under the visual servoing framework the docking behavior in our case becomes a more complex task due to the facts that: first, the state variables $\{P, \alpha, \beta\}$ for robot control are estimated by the visual servoing variables tilt angle a_{tilt}, pan angle a_{pan}, and the slope angle of the table edge a_{edge} for camera control, thus synchronizing and stabilizing the movement between the robot and the camera makes the robot controller no longer linear (especially in the goal pose) because of the dependent time lag and the momentum of the robot and camera control [7], [18]; second, the gripper on the PeopleBot robot only has 1-DOF for its up-down movement. Therefore precision control for positioning the robot to the goal pose is needed in order for the gripper to be able to execute the grasping action.

We will describe how to estimate the state variables $\{P, \alpha, \beta\}$ from object tracking and visual servoing in the next section in order to formulate a prior knowledge controller by equation 15 for our Q-RAN learning system.

B. Object Tracking and visual servoing

In our experiment, object tracking and edge detection of the table is relatively easy since the experimental setup is simplified with special colors for the object and table edge. As can be seen in Fig. 3, the green can is represented by its blob center $\{x_o, y_o\}$ in the image plane, calculated using a simple green-color threshold filter and the median $x-$ and $y-$ coordinates of the selected pixels. The slope angle a_{edg} of the table edge with respect to the robot's local frame is approximated by least-squares regression in which all N_r red pixels $\{x_{ri}, y_{ri}\}_{i=1}^{N_r}$ are filtered out from the image, then the red stripe is modeled as $Y_r = a_r + b_r X_r$, where

$$b_r = \frac{N_r \sum x_{ri} y_{ri} - \sum x_{ri} \sum y_{ri}}{N_r \sum x_{ri}^2 - (\sum x_{ri})^2}, \qquad (16)$$

$$a_r = \bar{y}_r - b_r \bar{x}_r, \qquad (17)$$

and $\bar{y}_r$ and $\bar{x}_r$ are the mean values of y_r and x_r.

The key idea of visual servoing for the pan-tilt camera control is to keep moving the camera so that the green can is in the center of the image. Let $\triangle Pan$ and $\triangle Tilt$ denote the relative pan and tilt angles of the camera so as to keep the green can in the center $(x_I = 160, y_I = 120)$ of the image, and $(dx, dy) = (x_o^{cur} - x_o^{pre}, y_o^{cur} - y_o^{pre})$ the difference between the current and previous positions of the green can in the image plane, as shown in the right column of Fig. 3. A simple PD-controller for servoing the camera is derived as,

$$\triangle Pan = K_{pp}(x_o^{cur} - x_I) + K_{dp} dx, \qquad (18)$$

$$\triangle Tilt = K_{pt}(y_o^{cur} - y_I) + K_{dt} dy, \qquad (19)$$

where the gains of the PD controller for camera control $K_{pp} = K_{pt} = 0.04$ and $K_{dp} = K_{dt} = 0.0015$ were found to give the best servoing results for the docking behavior.

Now that the servoing variables $\{a_{tilt}, a_{pan}, a_{edge}\}$ have been obtained from the visual servoing process, the state variables $\{P, \alpha, \beta\}$ can be estimated as $P = 80 - a_{tilt}$, $\alpha = a_{pan}$, and $\beta = a_{edge} = \arctan b_r$. The linear controller in equation 15 with the gains $K_p = 1.0$, $K_\alpha = 0.25$, and $K_\alpha = 0.35$ can then be used as a prior knowledge controller for rough control of the robot in our Q-RAN learning system. Note that in the above tracking and visual servoing procedures: (1) the image is processed in the HSV color space, (2) state estimation and visual servoing are carried out in the robot's local frame of reference, thus avoiding a global reference frame and odometry localization, (3) no calibration of the camera is needed.

C. Training the Q-RAN System

For the docking behavior in our experiments, the input vector $\boldsymbol{x}$ to the Q-RAN learning system is constructed from two state variables $\{\alpha, \beta\}$ plus the robot's rotational velocity v_{rot} that takes two discrete actions $u \in \{-8, 8\}$ (degree/s), thus resulting in $\boldsymbol{x} = [\alpha, \beta, u]^T$. The Q-RAN learning system is used to learn the control policy for the robot's rotational velocity v_{rot}. Note that robot's translational velocity (mm/s) is given by $v_{trans} = K_p P$, but P is not related to the odometry in the global frame since it is estimated by $P = 80 - a_{tilt}$ from the visual servoing process.

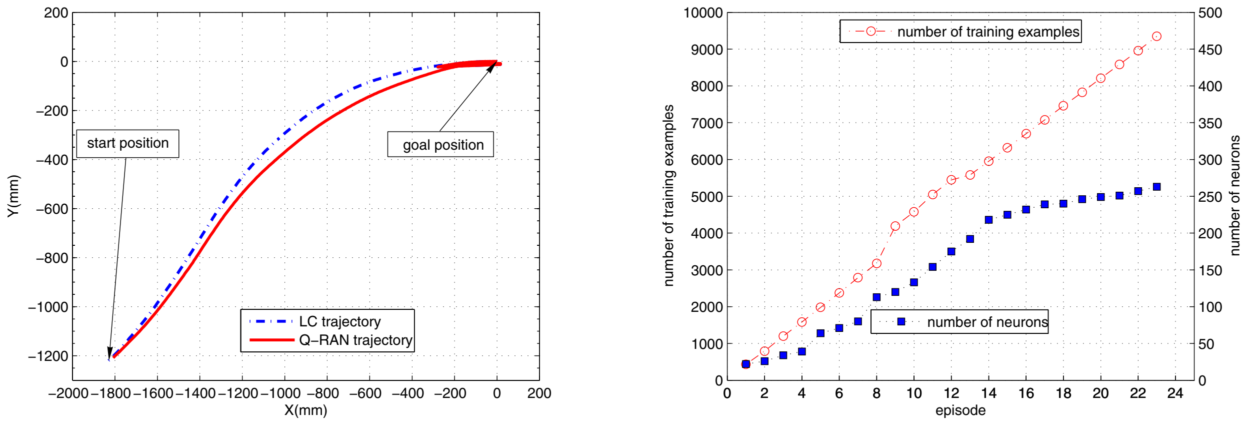
To train the Q-RAN learning system, all elements of $\boldsymbol{x}$ are normalized to the interval of $[0, 1]$; and the parameters of the Q-RAN learning system are set as in table I:

TABLE I

PARAMETERS CONFIGURATION FOR THE Q-RAN LEARNING SYSTEM

RAN Parameters Initialization:	$\eta_0 = 0.3$, $\tau = 50$, $\{d_{min}, d_{max}\} = \{0.07, 0.7\}$ $\epsilon = 0.2$, $\rho = 2.67$, $\kappa = 0.87$, $t_{max} = 1000$
Q-Learning	$\gamma = 0.99$, $\varepsilon = 0.2$
Goal State:	$\{P, \alpha, \beta\} = \{0, 0, 0\}$
Failure State:	$\lvert\alpha\rvert > 80$, or $\lvert\beta\rvert > 60$,or $t > t_{max}$

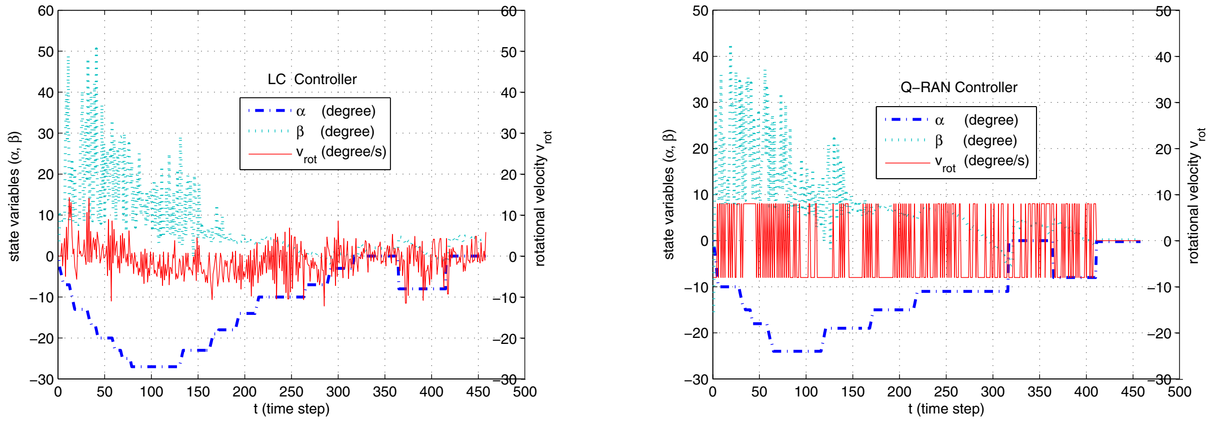
The reward function for Q-learning is given as follows: if the robot reaches the *goal state*, set the reward $r = +1.0$; if



(a) Trajectories of the linear and Q-RAN controllers

(b) Number of neuron and training examples vs. episode

Fig. 4. Left: trajectories comparison of two controllers. Right: neurons growing trends during the learning process of Q-RAN



(a) Linear controller

(b) Q-RAN controller

Fig. 5. Profile of the state variables (α, β) and rotation velocity v_{rot} for the two controllers

the robot reaches a *failure state*, that is, the robot either moves out of the state space or runs out of the time, set the reward $r = -0.2$; otherwise, set the reward $r = -0.001$ per step.

Based on the above configurations, an episode by episode training procedure for our Q-RAN learning system is summarized as follows:

(1) Estimate state variables $\{P, \alpha, \beta\}$ using object tracking and servoing as in section IV-B, and form the input vector $\boldsymbol{x}$,

(2) If the goal or a failure state reached, this episode is finished, randomly move the robot backward to a new starting position, and goto step **(1)** to start a new episode training,

(3) Else conduct Q-RAN's step 1 through step 11 as in section III, and set time step $t = t + 1$, goto step **(1)**.

V. Results and Discussion

In this section we discuss the performance of the Q-RAN learning system and give some observations obtained from our experiments as follows:

A. Training Q-RAN in Online Learning Mode

Using the epoch by epoch training procedure described in the preceding section, the Q-RAN learning system with the embedded linear controller successfully learned the docking behavior in completely online learning mode on the real robot. Specifically, the Q-RAN successfully learned the control policy for docking after 23 episodes in an experimental run (approx. 45 minutes), resulting in 263 neurons in Q-RAN's hidden layer (see Fig. 4(b).)

To compare the resulting Q-RAN controller with the linear controller, we conducted 10 trials of both controllers at a distance of approx. $2m$ away from the goal position. Fig 4 and 5 show several aspects of Q-RAN for the docking behavior:

Fig. 4(a) shows that the trajectory obtained with the Q-RAN controller (solid red line) is straighter than that of the linear controller (dash-dot green line), corresponding to a better approximation to the time-optimal behavior. More importantly, the Q-RAN controller succeeded in all 10 trials with an average of 405 ± 12 time steps per episode, compared to 8 successes of the linear controller with an average of 458 ± 14 time steps per episode.

A comparison of the Q-RAN controller with and without bootstrapping is out of the question here since random exploration on the real robot would require thousands of episodes. In fact, we tried the Q-RAN learning system without bootstrapping on the real robot, and it took 30 episodes (approx. 1 hour) for the robot to find the goal state for the first time during exploration. Obviously it would take much longer for the robot to find the optimal control policy.

Fig. 4(b) shows two aspects of the Q-RAN learning in the training process. On one hand, only 263 neurons (dot-square green line) were generated for some 10000 training examples (dash-circle red line), demonstrating Q-RAN's ability for generalization, thus avoiding the high storage requirements of memory-based approaches such as locally weighted learning (LWR) [1]. On the other hand, we notice that Q-RAN did not stop growing during the learning process, i.e. the network did not converge to a global optimum in terms of mean-squared error. However, the resulting Q-RAN controller did successfully control the robot for the docking behavior in our experiments. Sutton and Barto pointed out this feature while using function approximators (e.g., ANNs) in reinforcement learning (p. 196 and p. 222 in [17]); Li and Duckett also observed similar results for a wall following behavior [6], in which they found that a successful control policy can be acquired as long as the Q-values for the different actions in the same state are different enough to enable choice of the optimal action by $\arg\max_{u\in A}(Q(\boldsymbol{s},u)$.

Fig. 5 shows the relationships between the state variables $\{\alpha,\beta\}$ and the robot's rotational velocity v_{rot}. Specifically, Fig. 5(a) shows that the linear controller fails in its terminal state. This can be seen over steps $415-450$ (approximately), where β and v_{rot} are still oscillating while α approaches zero as the robot approaches the table, with its gripper over the can. This phenomenon is called "chattering" in control engineering.

By contrast, Fig. 5(b) shows that after time step 415, both the states (α,β) and the rotational velocity v_{rot} approach zero, meaning that the resulting Q-RAN controller successfully recognizes the goal state (where it receives the maximum reward in each successful training episode) while α and β reach zero simultaneously, thus resulting in a stable behavior (i.e., $v_{rot}=0$ when the goal state is reached).

In addition, we note that the Q-RAN controller is found to give the best results for docking when the rotational velocity v_{rot} is determined as a bang-bang controller, i.e., the action space includes only two actions $\{-8,8\}$. Large action spaces (e.g., $u\in\{$-8, -4, 0, 4, 8$\}$) were also tested in our experiments but did not outperform the bang-bang controller.

B. Training Q-RAN in a Layered Learning Architecture

As can be seen in Fig 4(a), while a successful Q-RAN controller for the docking behavior was acquired after 23 episodes of training, it does not significantly improve the trajectory in the sense of time-optimal behavior, compared to the linear controller. Therefore it is beneficial to further improve the Q-RAN controller's trajectory. To do this, we propose a layered learning architecture for training our Q-RAN system as shown in Fig. 6, which is actually inspired by Sharkey's work [14] in which he viewed the prior knowledge controller as an innate controller from a biological point of view, and experimentally verified that the supervised learning with ANNs could significantly improve the innate controller for obstacle avoidance and goal-finding behaviors.

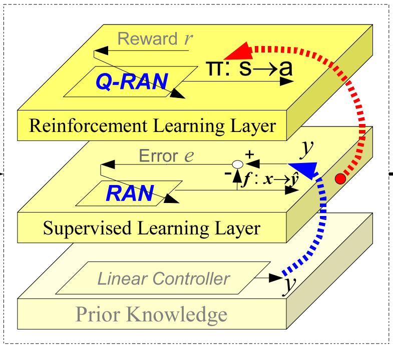


Fig. 6. The layered learning architecture of the Q-RAN learning system

In this layered learning architecture, the linear controller is first improved by the supervised learning layer with a RAN network. In the current implementation, this is done by offline learning. That is, the states (α,β) and the rotational velocity v_{rot} are recorded while the robot is guided by the linear controller for the docking behavior. Then, the collected data (α,β) are used as the inputs and v_{rot} as the desired output (i.e., the teaching signals) to train the RAN network. After training, the resulting RAN controller takes the states (α,β) as the inputs and calculates the rotational velocity as its output for the robot control.

Thereafter, the reinforcement learning layer with the Q-RAN takes over the training process in completely online mode using the RAN controller generated by the supervised learning layer as the prior knowledge controller. That is, the resulting RAN controller for rotational velocity v_{rot}, along with $v_{trans}=K_pP$ for the translational velocity, is used to control the robot in step 2 of the Q-RAN learning algorithm in section III. In the learning process of this layer, the RAN network takes $\boldsymbol{x}=[\alpha,\beta,u]^T$ as its inputs, and takes Q^{target} (which is generated by Q-learning, see Fig. 1) as its desired output (i.e., the teaching signals) for its online training.

Note that the RAN network is used in both the supervised and reinforcement learning layers, but its functionalities are different for each layer. In the supervised layer, the output of the RAN is the rotational velocity v_{rot} that can be directly sent to the robot as the motor command signal. By contrast, the output of the RAN in the reinforcement layer is the Q-value for the control policy π to choose the optimal action a at state s.

To evaluate the layered learning, we compared three controllers: the linear controller, the RAN controller bootstrapped by the linear controller, and the Q-RAN controller bootstrapped by the RAN controller. We performed 10 episodes starting at a longer distance of $4m$ from the goal position.

Fig. 7 shows that the linear controller (dash-dot blue line) is improved by the supervised learning layer with the RAN controller (dash dark line), and the RAN controller is further improved by the reinforcement learning layer with Q-RAN controller (solid red line) in the sense of the reduction of the

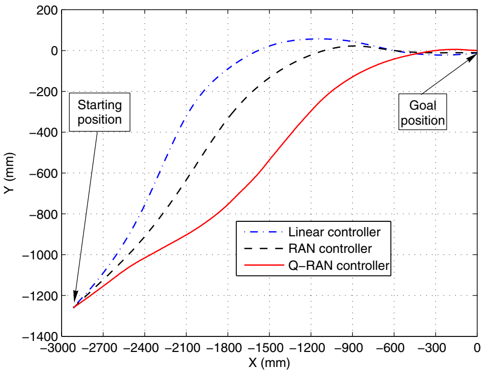


Fig. 7. Trajectories comparison for three different controllers

distance and the improvement of the trajectory overshoot. Note that again, the resulting Q-RAN controller succeeds in all 10 trials with an average of 518 ± 19 time steps per episode, compared to only 7 successes of the linear controller with an average of 685 ± 30 time steps per episode, and 9 successes of the RAN controller with an average of 618 ± 28 time steps per episode due to the the trajectory overshoots.

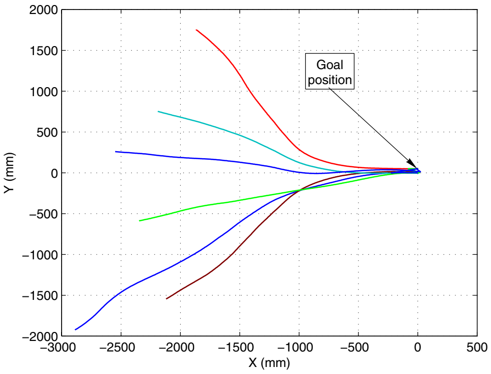


Fig. 8. Some example trajectories obtained with layered Q-RAN learning

Finally, Fig. 8 shows some sample trajectories of the Q-RAN controller resulting from our layered learning architecture, demonstrating that the robot starts at different starting poses and successfully fulfills the docking behavior.

VI. Conclusion

In this paper we investigated a Q-RAN learning system that is easy to use because the RAN's automatic growing mechanism simplifies the design process of the neural network structure and parameters. Our learning system can be speeded up in two ways: (1) training with an embedded prior knowledge controller in a complete online learning manner, (2) training in a layered learning architecture in which the supervised learning and the reinforcement learning are integrated with the same function approximator, a resource allocating network (RAN), thus allowing further refinement of the required behavior. We successfully applied the Q-RAN learning system for acquiring a vision-based docking behavior by a mobile robot.

Due to the fact that the RAN network can automatically adapt its network structure depending on the complexity of the required behavior, the task need not be known beforehand. This adheres to the emphasis of developmental robotics on "task non-specific" learning [21]. We are currently extending the Q-RAN learning algorithm to other behaviors to demonstrate the task non-specific nature.

References

[1] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(4):76–113, 1997.

[2] J. Bruske, I. Ahrns, and G. Sommer. An integrated architecture for learning of reactive behaviors based on dynamic cell structures. *Robotics and Autonomous Systems*, 22(2):87–101, 1998.

[3] K. R. Dixon, R. J. Malak, and P. K. Khosla. Incorporating prior knowledge and previously learned information into reinforcement learning agents. Technical report, Carnegie Mellon University, 2000.

[4] C. Gaskett. *Q-Learning for Robot Control*. PhD thesis, The Australian National University, 2002.

[5] G. B. Huang, P. Saratchandran, and N. Sundararajan. An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. *IEEE Trans. On System, Man, And Cybernetics–Part B: Cybernetics.*, 34(6):2284–2292, Dec. 2004.

[6] J. Li and T. Duckett. Q-learning with a growing RBF network for behavior learning in mobile robotics. In *Proceedings of the IASTED International Conference on Robotics and Applications (RA 2005)*, Cambridge, USA, Nov. 2005.

[7] I. R. Manchester and A. V. Savkin. Vision-based docking for biomimetic wheeled robots. In *16th IFAC world congress*, Prague, Czech Republic, July 2005.

[8] T. Martínez-Marín and T. Duckett. Fast reinforcement learning for vision-guided mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA 2005)*, Barcelona, Spain, 2005.

[9] D. L. Moreno, C. V. Regueiro, R. Iglesias, and S. Barro. Using prior knowledge to improve reinforcement learning in mobile robotics. In *Towards Autonomous Robotics Systems (TAROS04)*, UK, 2004.

[10] J. Platt. A resource allocating network for function interpolation. *Neural Computa.*, 3:213–225, 1991.

[11] B. Ratitch and D. Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *ECML-2004*, pages 347–358, 2004.

[12] F. Rivest and D. Precup. Combining TD-learning with cascade-correlation networks. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*. Washington DC, 2003.

[13] J. M. Santos and C. Touzet. Exploration tuned reinforcement function. *Neurocomputing*, 28(1-3):93–105, 1999.

[14] N. E. Sharkey. Learning from innate behaviors: a quantitative evaluation of neural network controllers. *Machine Learning*, 31:115–139, 1998.

[15] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, Cambridge, Massachusetts, 2004.

[16] W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *International Conference on Robotics and Automation*, May 11-15 2002.

[17] R. S. Sutton and A. Barto. *Reinforcement Learning, an introduction.* MIT Press, 1998.

[18] D. P. Tsakiris, P. Rives, and C. Samson. Extending visual servoing techniques to nonholonomic mobile robots. In G. Hager, D. Kriegman, and S. Morse, editors, *The Conference of Vision and Control, Lecture Notes in Control and Information Systems*. Springer-Verlag, 1998.

[19] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

[20] C. Weber, S. Wermter, and A. Zochios. Robot docking with neural vision and reinforcement. *Knowledge-Based Systems*, 17:165–172, 2004.

[21] J. Weng. Developmental robotics: Theory and experiments. *International Journal of Humanoid Robotics*, 1(2):199–236, 2004.

[22] R. J. Williams. Simple statistical gradient-following algorithm for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.