# Reinforcement Learning for a Vision Based Mobile Robot

Chris Gaskett, Luke Fletcher and Alexander Zelinsky

Robotic Systems Laboratory
Department of Systems Engineering, RSISE
The Australian National University
Canberra, ACT 0200 Australia
[cg|luke|alex]@syseng.anu.edu.au
http://syseng.anu.edu.au/rsl

**Abstract**

*Reinforcement learning systems improve behaviour based on scalar rewards from a critic. In this work vision based behaviours, servoing and wandering, are learned through a $\mathcal{Q}$-learning method which handles continuous states and actions. There is no requirement for camera calibration, an actuator model, or a knowledgeable teacher. Learning through observing the actions of other behaviours improves learning speed. Experiments were performed on a mobile robot using a real-time vision system.*

## 1 Introduction

Collision free wandering and visual servoing are building blocks for purposeful robot behaviours such as foraging, target pursuit and landmark based navigation. Visual servoing consists of moving some part of a robot to a desired position using visual feedback [15]. Wandering is an environment exploration behaviour [6].

In this work we demonstrate real-time learning of wandering and servoing on a real robot. Learning eliminates the calibration process and leads to flexible behaviour.

Reinforcement learning systems improve behaviour by learning to act in a way that brings rewards. A continuous state and action reinforcement learning system can generate motor commands which vary smoothly with the measured state. We also demonstrate that the learning system can develop through observing other behaviours—servoing is partly learned by observing the actions of the wandering behaviour, wandering is partly learned by observing the actions of the servoing behaviour.


Figure 1: A camera view during the wandering behaviour. Large squares represent detected obstacles.

## 2 Robot System Architecture

Our platform for research is a Nomad 200 with a Sony EVI-D30 colour camera. The camera points forward and downward from the robot (figure 2). The Nomad 200 is capable of forward-backward translation and rotation.

The camera signal is processed using a Fujitsu colour tracking vision card on-board the Nomad. The card is capable of performing around 200, eight by eight Sum of Absolute Difference (SAD) correlations per frame (at a frame rate of 30Hz).

The system architecture is based on the behaviour based system reported by Cheng and Zelinsky [6]. For these experiments the system has been simplified to two behaviours: *wandering* and *target pursuit*. Fig-

ure 3 shows an augmented finite state machine [4] representation of the behaviour based system.

The purpose of the wandering behaviour is to keep the robot moving without colliding with obstacles. In this system free space is detected by looking for carpet. A grid of 5×7 correlations are performed across the image space against a pre-loaded image of the carpet. The result is a matrix indicating the likelihood that regions ahead of the robot are carpet. The camera view in figure 1 shows smaller squares for regions which are likely to be carpet.

The target pursuit behaviour performs visual servoing to move the robot toward an 'interesting' object. Instead of using a pre-loaded template, an object is identified as interesting if it is not carpet but is surrounded by carpet. When an interesting object is identified the target pursuit behaviour dominates and servoing to the target begins. If the target is lost wandering resumes. Target pursuit together with wandering create a foraging behaviour.

In [6] both wandering and visual servoing employ a trigonometric model and a PID controller to translate the input 2d image coordinates into resultant translational and rotational velocities. Camera calibration is required, attitude and position relative to the floor must be measured.

In previous work the visual servoing behaviour was learned through reinforcement learning [7]. In this work both the wandering and visual servoing behaviours are learned. The camera calibration process is not required.

Figure 2: The Nomad 200 with colour camera.

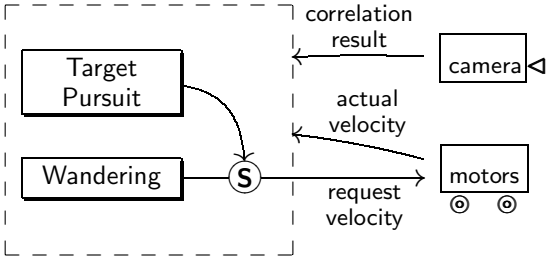



Figure 3: System Behaviour Model. Target pursuit dominates wandering when a target has been identified. Both behaviours use visual and motor velocity data.

## 3 Reinforcement Learning

A learning system is required which can form a mapping between state, including visual information, and actuator commands. A *supervised* learning approach would require a model of 'good behaviour' from a teacher—performance would be limited by the ability of this teacher. *Reinforcement* learning requires only a *critic*, that gives scalar rewards (or punishments) based on behaviour. An introduction to reinforcement learning methods is given in [20]. Providing a critic only requires that we have some measure of whether a task is being achieved, we do not need to know *how* to achieve the task. The reward signal need not be given immediately when an action is performed, as the true effect of an action can manifest itself after some time. Behaviour improves based on knowledge of which actions led to rewards and punishments. In this way, both good and bad experiences are a valuable part of the learning process. One popular reinforcement learning method, $\mathcal{Q}$-learning [24], is particularly flexible in this sense because it can learn from actions which it did not itself suggest, such as those from another controller, or historical data. This ability is often called exploration-insensitivity, we prefer the term policy-insensitivity.

$\mathcal{Q}$-learning works by incrementally updating the expected values of actions in states. For every possible state, every possible action is assigned a value which is a function of both the immediate reward for taking that action and the expected reward in the future based on the new state that is the result of taking that action. This is expressed by the one-step $\mathcal{Q}$-update equation:

$$\Delta\mathcal{Q}\left(\vec{x},\vec{u}\right) = \alpha\left[R + \gamma \max_{\vec{u}_{t+1}} \mathcal{Q}\left(\vec{x}_{t+1},\vec{u}_{t+1}\right) - \mathcal{Q}\left(\vec{x},\vec{u}\right)\right] \quad (1)$$

where $\mathcal{Q}$ is the expected value of performing action $\vec{u}$ in state $\vec{x}$, $R$ is the reward, $\alpha$ is a learning rate which controls convergence and $\gamma$ is the discount factor. The discount factor makes rewards earned earlier more valuable than those received later. The $\mathcal{Q}$-values implicitly describe a controller—measure the state, then choose the action with the highest $\mathcal{Q}$.

### 3.1 Continuous States and Actions

$\mathcal{Q}$-learning methods are best understood in the discrete case in which the state and the actions are symbolic rather than numerical (or continuous). Where real sensors, and commands to motors are concerned this leads to several problems: state aliasing, poor scaling with the number of states and actions, poor generalisation and coarse control. Several continuous state and action $\mathcal{Q}$-learning methods are briefly described in our earlier work [8]. Other methods are described in [21, 16, 3].

We use a continuous state, continuous action reinforcement learning algorithm based on an artificial neural network combined with an interpolator. This approach was investigated in simulation [9].

The combination of neural network and interpolator holds the $\mathcal{Q}$-values for all actions in all states. The input to the neural network is the state ($\vec{x}$), the output is a set of real valued actions ($\vec{u}$) and their values ($\vec{q}$) which is a sample of the $\mathcal{Q}$-function. The interpolator generalises between these actions.

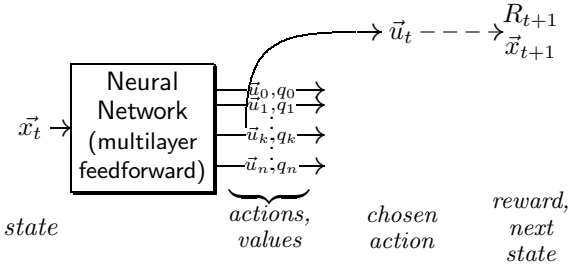
Baird and Klopf describe a suitable interpolation scheme called 'wire-fitting' [2]. In their work they combine the wire-fitter with a memory based reinforcement learning scheme, rather than a neural network. The wire-fitting function is a moving least squares interpolator, closely related to Shepard's function [17]. Each 'wire' is a combination of an action vector, $\vec{u}$, and its expected value, $q$, which is a sample of the $\mathcal{Q}$-function. The wire-fitting function is:

$$\mathcal{Q}(\vec{x},\vec{u}) = \lim_{\epsilon\to 0^+} \frac{\sum_{i=0}^{n} \frac{q_i(\vec{x})}{\|\vec{u}-\vec{u}_i(\vec{x})\|^2 + c[\max_j(q_j(\vec{x})) - q_i(\vec{x})] + \epsilon}}{\sum_{i=0}^{n} \frac{1}{\|\vec{u}-\vec{u}_i(\vec{x})\|^2 + c[\max_j(q_j(\vec{x})) - q_i(\vec{x})] + \epsilon}} \quad (2)$$

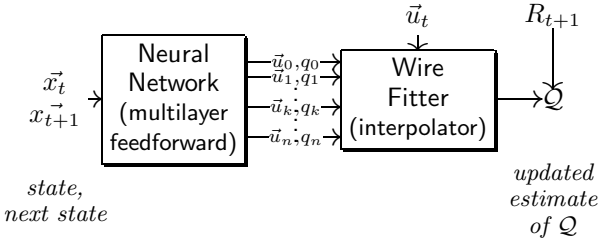
where $i$ is the wire number, $n$ is the total number of wires, $\vec{x}$ is the state vector, $\vec{u}_i(\vec{x})$ is the $i$th action vector, $q_i(\vec{x})$ is the value of the $i$th action vector, $\vec{u}$ is the action vector to be evaluated, $c$ is a small smoothing factor and $\epsilon$ avoids division by zero. The dimensionality of the action vectors $\vec{u}$ and $\vec{u}_i$ is the number of continuous variables in the action.

The wire-fitting function has several properties which make it a useful interpolator for implementing $\mathcal{Q}$-learning. Updates to the $\mathcal{Q}$-value (1) require $\max_{\vec{u}} \mathcal{Q}(\vec{x},\vec{u})$ which can be calculated quickly with the wire-fitting function. $\text{argmax}_{\vec{u}} \mathcal{Q}(\vec{x},\vec{u})$ can also be calculated quickly. This is needed when choosing an action to carry out. A property of this interpolator is that the highest interpolated value always coincides with the highest valued interpolation point, so the action with the highest value is always one of the the input actions. When choosing an action it is sufficient to propagate the state through the neural network, then compare the output $q$ to find the best action. The

1. In real time, feed the state into the neural network. Carry out the action with the highest $q$. Store the resulting change in state.




2. Calculate a new estimate of $\mathcal{Q}$ from the current value, the reward and the value of the next state. This can be done when convenient.




3. Calculate new values of $\vec{u}$ and $\vec{q}$ to produce the new value of $\mathcal{Q}$. Train the neural network to output the new $\vec{u}$ and $\vec{q}$. This can be done when convenient.

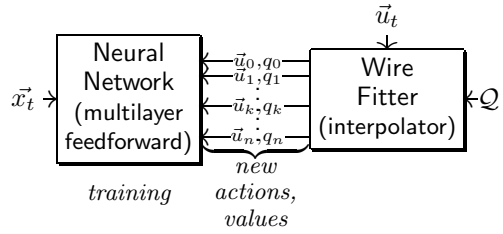



Figure 4: The training procedure. Step one is done in real time, the second and third can be done opportunistically.

wire-fitter is not required at this stage, the only calculation is a forward pass through the neural network. Wire-fitting also works with many dimensional scattered data while remaining computationally tractable; no inversion of matrices is required. Interpolation is local, only near wires influence the value of $\mathcal{Q}$. Areas far from all wires have a value which is the average of $\vec{q}$, wild extrapolations do not occur. It does not suffer from oscillations, unlike most polynomial schemes.

Importantly, partial derivatives in terms of each $q$ and $\vec{u}$ of each point can be quickly calculated [8]. These partial derivatives allow error in the output of the $\mathcal{Q}$-function to be propagated to the neural network according to the chain rule.

The training procedure is shown in figure 4. Training of the single hidden layer, feedforward neural network is done through incremental backpropagation. The learning rate is kept constant throughout. As suggested in [24], experiences are buffered and replayed repeatedly as if they are re-experienced.

### 3.2 Advantage Learning

A problem in $\mathcal{Q}$-learning is that a single suboptimal action may not prevent a high value action from being carried out at the next time step—the value of actions in a particular state can be very similar, as the value of the action in the next time step will be carried back. As the $\mathcal{Q}$-value is only approximated for continuous states and actions it is likely that most of the approximation power will be used representing the values of the states rather than actions in states. The relative values of actions will be poorly represented, resulting in an unsatisfactory controller. This is compounded as the time intervals between control actions get smaller. The 'state action deviation problem' is closely related to this problem [1].

Advantage Learning [10] addresses the issue of action similarity by emphasising the differences in value between the actions. In advantage learning the value of the optimal action is the same as for $\mathcal{Q}$-learning, but the lesser value of non-optimal actions is emphasised by a scaling factor ($k \propto \Delta t$). This makes a more efficient use of the approximation resources available. Equation 3 is the advantage learning update. The quantity $\mathcal{A}$ is analogous to $\mathcal{Q}$ in (1).

$$\Delta\mathcal{A}(\vec{x},\vec{u}) = \alpha[\tfrac{1}{k}\left(R + \gamma \max_{\vec{u}_{t+1}} \mathcal{A}(\vec{x}_{t+1},\vec{u}_{t+1})\right) + \left(1-\tfrac{1}{k}\right)\max_{\vec{u}_{t+1}} \mathcal{A}(\vec{x}_t,\vec{u}_t) - \mathcal{A}(\vec{x},\vec{u})] \quad (3)$$

In our simulation experiments Advantage Learning improved convergence speed and reliability [8].

## 4 Learning to Wander

The purpose of the wandering behaviour is to explore an environment without colliding with obstacles. The hardwired algorithm uses a matrix of correlations with a carpet template to guide the movement of the robot: The robot moves in the direction of the longest uninterrupted strip of carpet or rotates if no carpet is detected [6].

We conjectured that successful wandering involves maximising the amount of carpet in view while maximising forward velocity. Table 1 shows the state, actions and reward for the reinforcement learning algorithm. The reward is a weighted sum of the components shown. Carpet nearer to the robot is weighted more heavily. Rewarding for forward movement also punishes backward motions, this makes blind reversing less desirable. The last component of the reward signal punishes for non-smooth motion which could be unpredictable to bystanders and eventually harm the robot.

**Wandering:**

**State** $C$: carpet match matrix
$t$, $r$: translational and rotational velocity of robot (measured by encoders)
$ts$, $rs$: smoothed trace of previous actions

**Action** $T$, $R$: target translational and rotational velocity of robot

**Reward** $-\sum C \cdot row$: maximise visible carpet
$+t$: move forward
$-\{(T-t)^2 + (R-r)^2\}$: smooth motion

Table 1: Reinforcement learning specification for wandering.

It is important that the state representation be as accurate and complete as possible. Early in testing, the velocity of the robot (measured with encoders) was not provided as state information. Learning failed in this case. Analysis of the data showed that because of the restricted acceleration available, the difference between the current and next state was negligible, being of the same order as the noise in the system. As a result the state information was simply filtered out—there was effectively no correlation between the command and the change in state.

It is difficult to obtain accurate data from the carpet matching process. Lighting varies throughout the corridors in the test area. The walls are a similar colour to the carpet. To increase reliability we add

the constraint that if a non-carpet region is detected the region above it cannot be carpet (see figure 1). We also smooth the carpet correlations with their neighbours horizontally and smooth between frames. These processing steps raise the quality of the state data to a standard at which learning is possible.

We found that it is necessary to provide a safety buffer during learning to prevent damage to the robot. The infra-red ring on the the Nomad is used to veto actions which would cause immediate collision. When a veto occurs there is an audible warning so that we are aware of intervention by the hardwired behaviour. The infra-red readings are not part of the state for the learning algorithm as this would make it unclear whether the vision system is being used.

At the beginning of the trial the robot's behaviour is random, but the initialisation of the neural network with small weights produces only slow movement. Unless the robot starts directly facing a wall, it quickly learns that forward velocity brings rewards. When it reaches a wall the robot finds that moving forward is not always good and adapts its behaviour accordingly. As learning continues the number of action vetoes is reduced, they still occur when the robot encounters an obstacle it cannot detect.

Figure 5 shows a path recorded while wandering. At the point marked 'start' learning begins. Movement becomes smoother and more consistent throughout the trial.

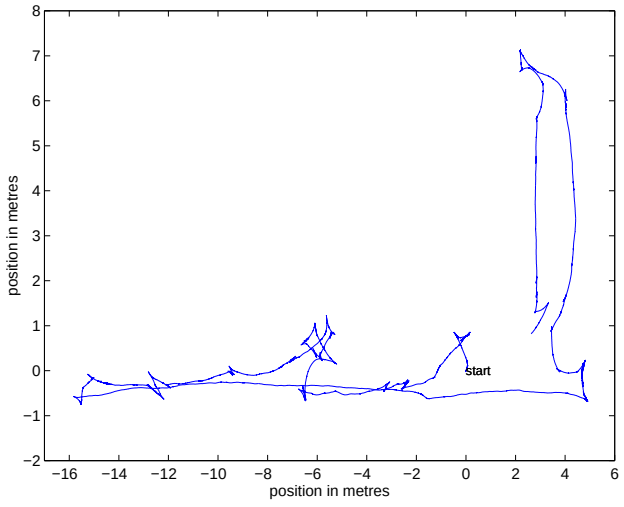



Figure 5: A path from a wandering experiment in an open plan office. Several pillars and a plant pot were avoided. Total trial time was 25 minutes, or 15637 actions. The trial ended when the robot clipped a pillar that was not within view and not detected by the infra-red 'bumper'.

## 5 Learned Visual Servoing

Visual servoing [15] is a useful capability for manipulators and mobile robots. Visual servoing requires the ability to track the position of some object using vision, and to control some effector based on feedback from the tracking.

Area correlation-based tracking is a process that locates objects between successive frames and hence can be used to gauge the effect of the robot's motion in image space. An image (template) is captured from a particular region of the image space and stored in a buffer. The template is then compared in the next video frame in the neighbourhood of its location in the previous frame. Various methods are used to measure the degree of similarity between the images [15, 6]. The difference in location between the template in the current frame and the best match of the template in the next frame forms a vector which indicates the motion of the target. To track an object, a template is captured, then starting from the original location, the motion vectors from each successive frame are added. Tracking failure is indicated when the measure of difference between the template and the closest match in the current image is too great [15].

There are two basic approaches to the control part of visual servoing: position based and image based. Both generally require some form of calibration.

In position based systems an error signal is defined in the robot's coordinate system. A model describing the relationship between visual coordinates and the robot's coordinate system is required. It is sometimes possible to learn this model [11, 19, 18, 5]. Such systems are suitable for servoing manipulator arms, where joint angles define the position of an effector.

In image based systems the error signal is defined in image space. The inverse of the image Jacobian is used to relate desired incremental change to the image to changes in actuator settings. It is possible to learn an approximation of this Jacobian [14] but this is complicated since it can vary with the state [25].

Our approach is to learn a direct mapping from image space and other state information to actuator command using reinforcement learning [7]. The same approach has been developed independently in [21]. Our method may not be able to achieve the performance of well calibrated systems but we certainly gain flexibility; if the camera is bumped or moved or the actuator configuration is changed the system still works.

Table 2 shows the state, actions and reward for the reinforcement learning algorithm for learned visual servoing. The reward function includes a term which punishes for energy use. This helps to reduce

**Visual Servoing:**

**State** $x$, $y$: pixels error to target
$\Delta x$, $\Delta y$: pixels velocity of target
$t$, $r$: translational and rotational velocity of robot (measured by encoders)

**Action** $T$, $R$: target translational and rotational velocity of robot

**Reward** $-x^2 - y^2$: movement to target
$-T^2 - R^2$: saving energy
$-\{(T-t)^2 + (R-r)^2\}$: smooth motion

Table 2: Reinforcement learning specification for visual servoing.

the robot's reaction to minor changes in the target position due to tracking noise. There is no punishment term for target velocity due to excessive noise in the target velocity estimation process.

The robot successfully learns to servo to the target, even when the camera is misaligned. When the camera is mounted facing straight ahead the learned behaviour is to turn toward the target while driving forward until the target is at the goal position. The robot slows down as it gets closer to the target. The robot also learns to reverse if the target is closer than the goal location. For the misaligned camera position the robot is often unable to turn the wheels in the direction of the target without moving the target into a worse location or losing the target completely. The learning system instead develops a zig-zagging method to move toward the target.

Learning from already working behaviours reduces the period of random behaviour which would require close supervision. In this experiment data gathered while wandering is provided to the learning to servo system. The wandering behaviour learns from the actions of the servoing behaviour in the same way. The $\mathcal{Q}$-learning system's policy-insensitivity allows it to learn from this data.

Visual servoing performance is adequate after 15 minutes of real time, or about 1500 tracking frames. Performance continues to improve over an hour of experimentation. The robot learns to servo to objects placed in any position in its visual field.

## 6 Relation to Other Work

A non-learning vision based physically grounded obstacle avoidance algorithm was presented in [13]. Visual servoing through reinforcement learning was demonstrated in [23]. Learned wall avoidance in a constructed environment with discrete actions is described in [22]. Asada has developed several vision based reinforcement learning systems, for example [1]. These systems have used discrete actions and adaptively discretised the state space. More recently a visual servoing system using reinforcement learning with continuous states and actions was presented in [21].

## 7 Conclusion

We have demonstrated wandering and servoing behaviours on a real mobile robot that are learned through trial and error using reinforcement learning. A continuous state and action $\mathcal{Q}$-learning system generated actions which varied smoothly with the measured state. Learning time is decreased through learning from other behaviours.

In future work we will move toward an active stereo vision system and development of more sophisticated behaviours.


### Acknowledgments

The behavioural system software is based on the work of Gordon Cheng [6] and uses correlation software developed by Jochen Heinzmann for real time face tracking [12].


## References


[1] M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda. Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In *Proc. of IAPR/IEEE Workshop on Visual Behaviors*, 1994.

[2] Leemon C. Baird and A. Harry Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright Laboratory, 1993.

[3] H. R. Berenji. Fuzzy Q-learning: a new approach for fuzzy dynamic programming. In *Proc. Third IEEE Int. Conf. on Fuzzy Systems*, NJ, 1994. IEEE Computer Press.

[4] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[5] J.L. Buessler, J.P. Urban, H. Kihl, and J. Gresser. A neural model for the visual servoing of a robotic manipulator. In *Proc. Symposium on Control, Optimization and Supervision, Computational Engineering in Systems Applications (CESA'96)*, Lille, France, 1996.

[6] G. Cheng and A. Zelinsky. Real-time visual behaviours for navigating a mobile robot. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '96)*, volume 2, Osaka , Japan, 1996.

[7] Chris Gaskett, Luke Fletcher, and Alexander Zelinsky. Reinforcement learning for visual servoing of a mobile robot. In *Proc. of the Australian Conference on Robotics and Automation (ACRA2000)*, 2000.

[8] Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Q-learning in continuous state and action spaces. In *Proc. of the 12th Australian Joint Conference on Artificial Intelligence*, Sydney, Australia, December 1999. Lecture Notes in Computer Science, Springer-Verlag.

[9] Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Reinforcement learning applied to the control of an autonomous underwater vehicle. In *Proc. of the Australian Conference on Robotics and Automation (AUCRA'99)*, 1999.

[10] Mance E. Harmon and Leemon C. Baird. Residual advantage learning applied to a differential game. In *Proc. of the International Conference on Neural Networks*, Washington D.C, 1995.

[11] H. Hashimoto, T. Kubota, M. Kudou, and F. Harashima. Self-organizing visual servo system based on neural networks. In *Proc. American Control Conference*, Boston, 1991.

[12] Jochen Heinzmann and Alexander Zelinsky. Robust real-time face tracking and gesture recognition. In *Proc. of the Int. Joint Conf. on Artificial Intelligence, IJCAI'97*, 1997.

[13] I. Horswill. Visual collision avoidance by segmentation. In *Intelligent Robots and Systems, (IROS'94)*, pages 902–909, 1994.

[14] K. Hosoda and M. Asada. Versatile visual servoing without knowledge of true jacobian. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, 1994.

[15] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, October 1996.

[16] M.I. Jordan and R.A. Jacobs. Learning to control an unstable system with forward modeling. In *Proc. Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990.

[17] Peter Lancaster and Kęstutis Šalkauskas. *Curve and Surface Fitting, an Introduction*. Academic Press, 1986.

[18] Wai Chau Lo. Robotic visual servo control. In *Proc. Twelfth International Conference on CAD/CAM, Robotics and Factories of the Future*, London, 1996.

[19] Jae Seock Park and Se Young Oh. Dynamic visual servo control of robot manipulators using neural networks. *Journal of the Korean Institute of Telematics and Electronics*, 29B(10), 1992.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Bradford Books, MIT, 1998.

[21] Y. Takahashi, M. Takeda, and M. Asada. Continuous valued Q-learning for vision-guided behavior. In *Proc. of IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 1999.

[22] K. Terada, H. Takeda, and T. Nishida. An acquisition of the relation between vision and action using self-organizing map and reinforcement learning. In *Proc. Second International Conference. Knowledge-Based Intelligent Electronic Systems, (KES'98)*, 1998.

[23] Sebastian Thrun. An approach to learning mobile robot navigation. *Robotics and Autonomous Systems*, 15(4):301–319, 1995.

[24] Christopher J. C. H. Watkins and Peter Dayan. Technical note: Q learning. *Machine Learning*, 8:279–292, 1992.

[25] Q.M.J. Wu and K. Stanley. Modular neural-visual servoing using a neuro-fuzzy decision network. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'97)*, Albuquerque, NM, 1997.