Reinforcement Learning for Visual Servoing of a Mobile Robot

Chris Gaskett, Luke Fletcher and Alexander Zelinsky
Robotic Systems Laboratory
Department of Systems Engineering
Research School of Information Sciences and Engineering
The Australian National University
Canberra, ACT 0200 Australia
[cg|luke|alex]@syseng.anu.edu.au
http://syseng.anu.edu.au/rs[1]

Abstract

A novel reinforcement learning algorithm is applied to a visual servoing task on a real mobile robot. There is no requirement for camera calibration, an actuator model or a knowledgeable teacher. The controller learns from a critic which gives a scalar reward. The learning algorithm handles continuously valued states and actions and can learn from good and bad experiences including data gathered while performing unrelated behaviours and from historical data. Experimental results are presented.

1 Introduction

Visual servoing consists of moving some part of a robot to a desired position using visual feedback [Hutchinson *et al.*, 1996]. It is a basic building block for purposeful robot behaviours such as foraging, target pursuit and landmark based navigation. Some degree of calibration is generally required to achieve visual servoing. This calibration can be a time consuming and error prone process.

In this work we show that reinforcement based learning can eliminate the calibration process and increase flexibility. The reinforcement learning based controller learns to act in a way that will bring rewards, in this case rewards are given for approaching the visual target and smoothly controlling the motors.

We demonstrate that a continuous state, continuous action, exploration insensitive reinforcement learning algorithm can learn in real time to visually servo a mobile robot to a target. Exploration insensitivity allows the algorithm to learn from a diverse range of experiences gathered from performing other behaviours. The generality of the learning algorithm makes it useful for behaviours apart from servoing.

Figure 1: The Nomad 200 with colour camera.

2 Visual Servoing

Visual servoing [Hutchinson *et al.*, 1996] is a useful capability for both manipulator arms 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 result 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 on the previous frame. Various methods can be used to measure the degree of similarity between the images [Hutchinson *et al.*, 1996; Cheng and Zelinsky, 1996]. The difference in location between the template in the current frame and the best match of the template in the next frame form 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. Failure of the tracking is indicated when the measure of difference between the template and the closest match in the current image is too great [Hutchinson *et al.*, 1996].

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 [Hashimoto *et al.*, 1991; Park and Oh, 1992; Lo, 1996; Buessler *et al.*, 1996]. Such systems are more 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 [Hosoda and Asada, 1994] but this is complicated because it varies with the state [Wu and Stanley, 1997].

Our approach is to learn a direct mapping from image space and other state information to the actuator command using reinforcement learning. The same approach has been developed independently in [Takahashi *et al.*, 1999]. Our method may not be able to achieve the performance of well calibrated systems but we certainly gain flexibility; if the camera is moved or the actuator configuration is changed the system still works.

3 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 1). 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 by Cheng and Zelinsky [Cheng and Zelinsky, 1996], shown in figure 2. The default behaviour is free space searching (wandering). A grid of 6×7 correlations are performed on the image space against a preloaded image of the carpet. The result is a binary matrix indicating 'carpet' or 'not carpet' regions ahead of the robot. The wandering behaviour moves the Nomad toward the regions of most carpet matches (see figure 3).

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

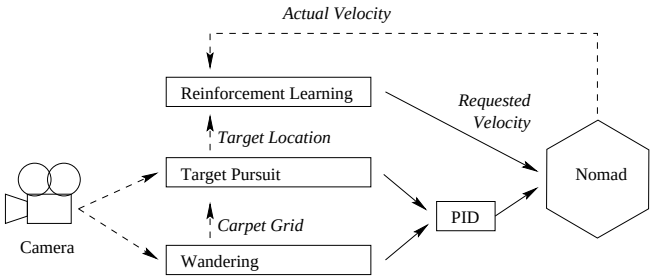


Figure 2: System Behaviour Model.

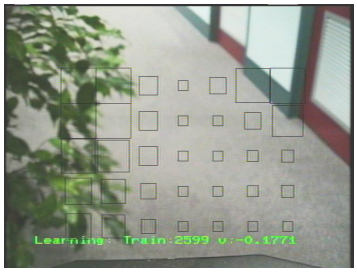


Figure 3: Typical camera view during the wandering behaviour. The size of the squares represents the degree of difference from the carpet template.

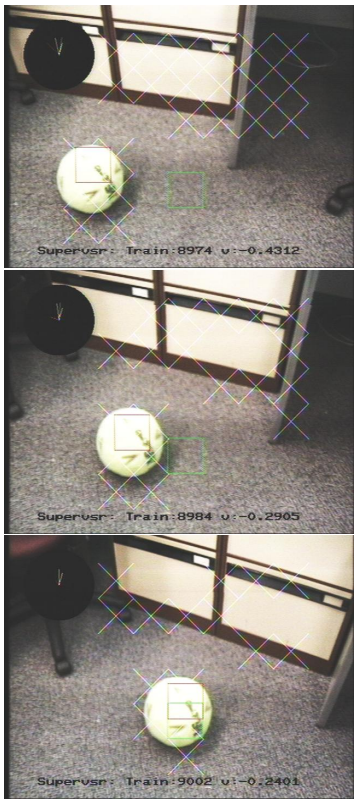


Figure 4: Visual servoing under the control of reinforcement learning.

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 this work we replace the hard wired target pursuit behaviour with a learned visual servoing behaviour. This behaviour generates translational and rotational velocities given the target and home coordinates in image space.

As we are looking at a 3D world through a monocular camera we make the assumption that all targets are close to floor level. Occasionally targets are tracked which are above floor level (i.e., on a desk). Objects which are fairly uniform in the vertical (i.e., trouser ledgs) occasionally cause the template tracking to slide upward. This erroneous data is regarded as a form of noise to the learning process.

4 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 [Sutton and Barto, 1998]. 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. In our case the reward is the negative of the distance between the tracked target's current position, and the desired position. Punishment is also given for use of energy and large changes in motor commands. 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 [Watkins and Dayan, 1992], 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. In our experiment, learning speed is improved by gathering data from another controller which is completely separate to the visual servoing task.

$\mathcal{Q}$-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:

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

where $\mathcal{Q}$ is the expected value of performing action $\vec{u}$ in state $\vec{x}$, R is the reward, α is a learning rate which controls convergence and γ 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 $\mathcal{Q}$.

4.1 Continuous States and Actions

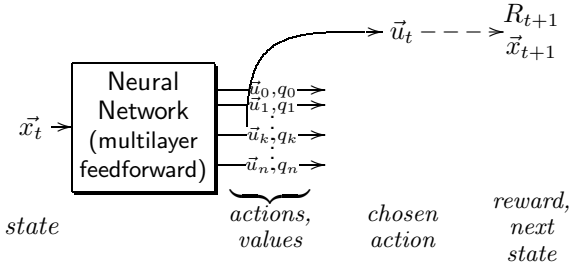
$\mathcal{Q}$-learning methods are best understood in the discrete case in which the state and the actions are symbolic rather than numerical (or continuous). Where real sensors, and commands to motors are concerned this leads to several problems: state aliasing, poor scaling with the number of states and actions, poor generalisation and coarse control. Several continuous state and action $\mathcal{Q}$-learning methods are briefly described in our earlier work [Gaskett *et al.*, 1999a]. Other methods are described in [Takahashi *et al.*, 1999; Jordan and Jacobs, 1990; Berenji, 1994].

We use a continuous state, continuous action reinforcement learning algorithm based on an artificial neural network combined with an interpolator. Previously we applied this approach to a simulation task [Gaskett *et al.*, 1999b]. 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.

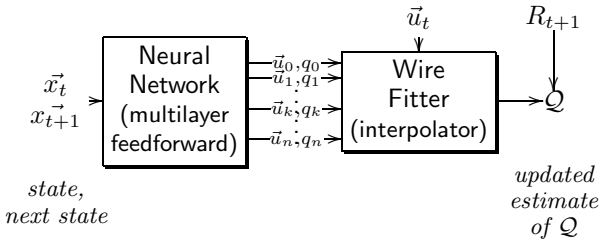
Baird and Klopf describe a suitable interpolation scheme called 'wire-fitting' [Baird and Klopf, 1993]. In their work they combine the wire-fitter with a memory based reinforcement learning scheme, rather than a neural network. The wire-fitting function is a moving least squares interpolator, closely related to Shepard's function [Lancaster and Šalkauskas, 1986]. 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:

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

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.

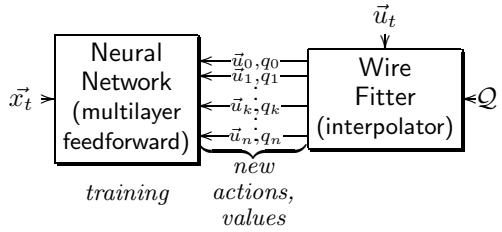


Figure 5: The learning procedure. Step 1 is done in real time, steps 2 and 3 can be done opportunistically.

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 ith action vector, $q_i(\vec{x})$ is the value of the ith action vector, $\vec{u}$ is the action vector to be evaluated, c is a small smoothing factor and ϵ 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. $\operatorname{argmax}_{\vec{u}} \mathcal{Q}(\vec{x}, \vec{u})$ 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 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 [Gaskett *et al.*, 1999a]. 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 5. Training of the single hidden layer, feedforward neural network is done through incremental backpropagation. The learning rate is kept constant throughout. As suggested in [Watkins and Dayan, 1992], experiences are buffered and replayed repeatedly as if they are re-experienced.

4.2 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.

Advantage Learning [Harmon and Baird, 1995] 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 simulation experiments Advantage Learning improved convergence speed and reliability [Gaskett *et al.*, 1999a].

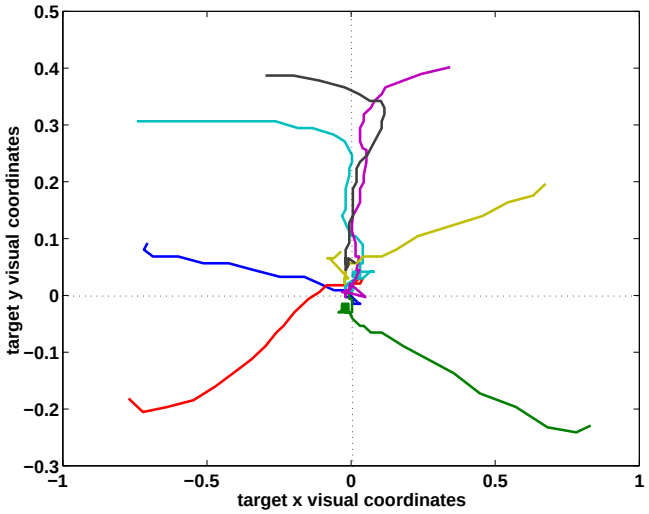


Figure 6: Example target trajectories.

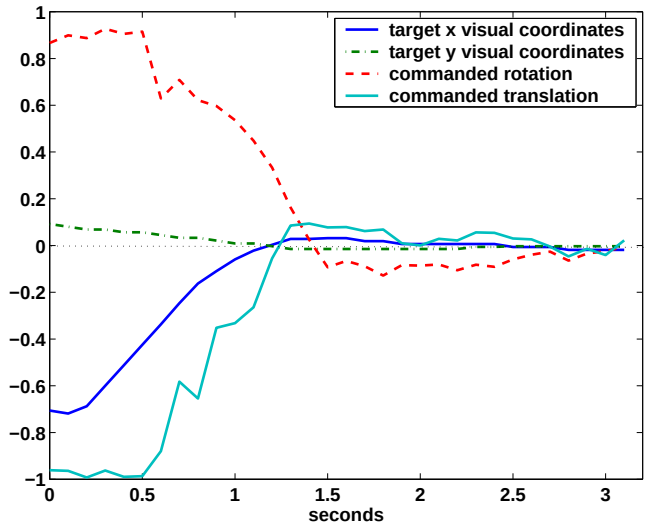


Figure 7: Example Step response and motor commands.

5 Experimental Results

The experimental system has been described in section 3. The task is for the robot to learn to visually servo a target object to the center lower half of the visual field. Figure 4 shows a sequence of three frames, one per second of a 10Hz video sequence. The target selected by the interest operator in this case is a soccer ball.

Table 1 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 reactions to minor changes in the target position due to tracking noise.

The learning data is organised into the form: state, action, next state, reward. Two hundred of these 'experiences' are gathered while wandering. The rest are gathered while attempting to servo to the target. The learning algorithm processes these experiences (between frames) and incrementally increases its ability to servo to the target. Figure 6 shows trajectories of targets in image space. Example step response and motor commands are shown in figure 7.

Visual Servoing:

State x, y: pixels error to target
Δx, Δy: pixels velocity of target
t, r: translational and rotational velocity of robot (measured by encoders)

Action T, R: commanded 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 1: Learning specification for visual servoing.

The robot successfully learns to servo to the target. The learned behaviour is to turn toward the target while driving forward until the target is at the goal position. The robot also learns to reverse if the target is closer than the goal location.

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 given 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.

Gathering data while performing other behaviours, such as wandering, is beneficial. Without this the robot needs a long period of learning to begin moving in any purposeful way. The servoing behaviour does more than just reuse parts of the built in wandering behaviour—the wandering behaviour never moves the robot backward, but the learned servoing behaviour does move the robot backward if the target is too close to the robot. The policy-insensitivity which allows this flexible learning is an important characteristic of $\mathcal{Q}$-learning systems for real world applications.

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.

The experiment was repeated with a grossly misaligned camera (20° roll, 45° yaw). This necessitated a zig-zagging strategy in order to reach the goal. Results were similar in terms of steps and learning time.

Conclusion

We have demonstrated a visual servoing behaviour for a real mobile robot which is learned through trial and error using reinforcement learning. No calibration is required. The distinctive features of the reinforcement learning algorithm used are its ability to use continuous states and actions, and its policy-insensitivity. In future work we will attempt to develop other learned behaviours.

Acknowledgements

The visual servoing software is based on the work of Gordon Cheng [Cheng and Zelinsky, 1996] and uses correlation software developed by Jochen Heinzmann for real time face tracking [Heinzmann and Zelinsky, 1997].

References

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

[Berenji, 1994] 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.

[Buessler *et al.*, 1996] 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.

[Cheng and Zelinsky, 1996] 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.

[Gaskett *et al.*, 1999a] 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.

[Gaskett *et al.*, 1999b] 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.

[Harmon and Baird, 1995] 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.

[Hashimoto *et al.*, 1991] 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.

[Heinzmann and Zelinsky, 1997] 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.

[Hosoda and Asada, 1994] 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.

[Hutchinson *et al.*, 1996] 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.

[Jordan and Jacobs, 1990] 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.

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

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

[Park and Oh, 1992] 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.

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

[Takahashi *et al.*, 1999] 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.

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

[Wu and Stanley, 1997] 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.