Computer Science Department of The University of Auckland
CITR at Tamaki Campus (http://www.citr.auckland.ac.nz)

CITR-TR-47 June 1999

Path-Tracking Control of a Non-Holonomic Car-Like Robot with Reinforcement Learning

Jacky Baltes [1], and Yuming Lin [1]

Abstract

The problem investigated in this paper is that of driving a car-like robot along a race track and the use of reinforcement learning to find a good control function. The reinforcement learner uses a case-based function approximator to extend the reinforcement learning paradigm to handle continuous states. The learned controller performs similar to the best control functions in both simulation and also in practical driving.

[1] CITR, Tamaki Campus, University of Auckland, Private Bag 92019

Path-Tracking Control Of Non-holonomic Car-Like Robot With Reinforcement Learning

Jacky Baltes

Yuming Lin

CITR, University of Auckland

Tamaki Campus

Auckland, New Zealand

Email: j.baltes@auckland.ac.nz, ylin045@student.auckland.ac.nz

May 13, 1999

Keywords: Mobile Robot, Nonholonomic Control, Reinforcement learning.

Abstract

The problem investigated in this paper is that of driving a car-like robot along a race track and the use of reinforcement learning to find a good control function. The reinforcement learner uses a case-based function approximator, to extend the reinforcement learning paradigm to handle continuous states. The learned controller performs similar to the best control functions in both simulation and also in practical driving.

Computer Science Depatrment of The University of Auckland
CITR at Tamaki Campus (http://www.citr.auckland.ac.nz)

CITR-TR-47 June 1999

Path-Tracking Control of a Non-Holonomic Car-Like Robot with Reinforcement Learning

Jacky Baltes [1], and Yuming Lin [1]

Abstract

The problem investigated in this paper is that of driving a car-like robot along a race track and the use of reinforcement learning to find a good control function. The reinforcement learner uses a case-based function approximator to extend the reinforcement learning paradigm to handle continuous states. The learned controller performs similar to the best control functions in both simulation and also in practical driving.

[1] CITR, Tamaki Campus, University of Auckland, Private Bag 92019

1 Introduction

The CITR at the University of Auckland has a mobile robotics lab, which hosts the Aucklandianapolis competition ([2]). The goal of the competition is to drive car-like (non-holonomic) robots five laps around a race track as quickly as possible. The cars are simple remote controlled toy cars with proportional steering and speed. A parallel port micro-controller based interface ([6]) allows us to control the cars (65 speed settings, 65 direction settings). Position and orientation information for the cars is provided by a video camera mounted on top of the playing field.

A non-holonomic path planner ([3]) creates a path for the car around the race track. The path contains only three different path segments: (a) straight lines, (b) maximum turns to the right, or (c) maximum turns to the left.

Some popular methods to control a non-holonomic mobile robot in such a path tracking problem include:

1. Feedback control as described by Alessandro and Giuseppe [4].

2. A Sliding-mode controller suggested in [1], which was used during initial trials for the Aucklandianapolis. This state of the art controller performed extremely well in simulation, but performed poorly in the real world. The motivation of this project was to improve on its performance in the real world, see section (5).

3. A Fuzzy logic controller [7] which currently holds the unofficial track record for the Aucklandianapolis. The fuzzy logic controller is able to drive a car twice as fast as the sliding mode controller mentioned above.

This paper describes another method based on dynamic programming, a

reinforcement learning controller. Because at the core of the reinforcement learner is a value function, called Q-value, it is also called Q-Learning ([8]).

The following section describes the kinematic model of the car-like vehicle, or just car for simplicity. The model is used throughout the paper. Section 3 gives a brief introduction to reinforcement learning. Section 4 describes a case-based function approximator, which is used as a representation of the value function in our implementation. Section 5 describes the results of our experiments using both simulation and practical driving. Section 6 concludes the paper.

2 Kinematic Model

In this research, we use a kinematic model, which is relative to the path. The controller knows the current position and orientation errors and the curvature of the path. However, the future path is not known. This model is appropriate in highly dynamic domains such as RoboCup.

The kinematic model is shown in figure 2. The car is at position (x,y) and is following a path with curvature R. The point ($\hat{x}$,$\hat{y}$) is the closest point on the path to point (x,y). The position error $\tilde{y}$ is the distance between points (x,y) and ($\hat{x}$,$\hat{y}$). $\hat{\theta}$ is the tangent of the path at the point ($\hat{x}$,$\hat{y}$), θ is the orientation of the car, $\tilde{\theta}$ is the orientation error of the car (that is, $\tilde{\theta} = \theta - \hat{\theta}$).

In our reinforcement learning paradigm, the current state of the system is defined by $\tilde{y}$,$\tilde{\theta}$ and the curvature of the path R (a 3-tuple). The input to the controller is the three tuple for the current state and the output are desired controls for speed and direction.

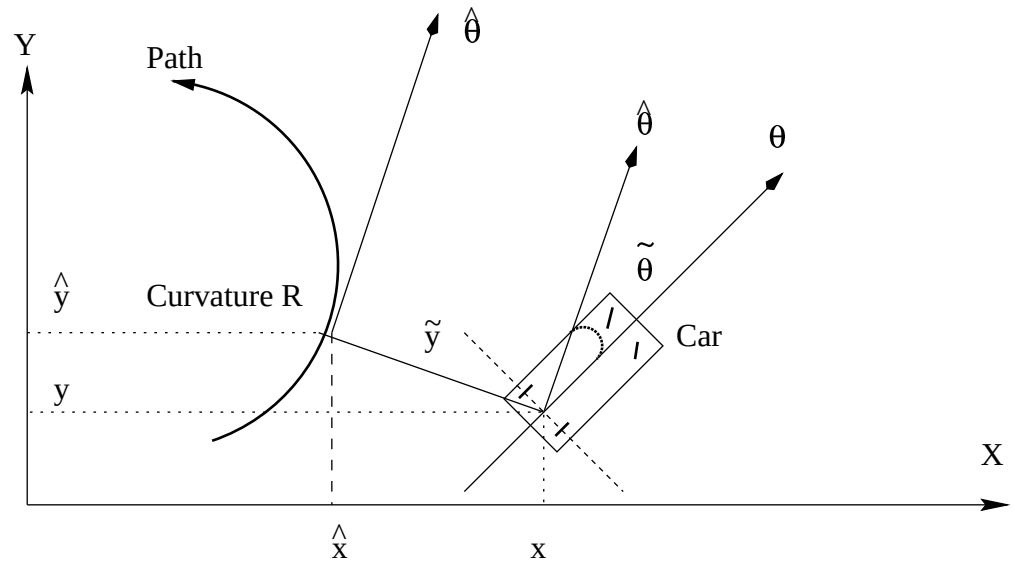

Figure 1: The Kinematic Model

3 Reinforcement Learning

This section gives a brief introduction to reinforcement learning. In reinforcement learning, agent and environment are the most important concepts. The agent (a remote-controlled car in our case) has a number of possible actions. The agent performs some actions in the environment (which is modeled through a set of states). In some states, the agent receives feedback about how good or bad a certain state is from the environment, a so called reward. The task of reinforcement learning is to find the optimal action so that the reward is maximized. In a control task, the reward is based on how well the agent tracked the given path.

At any time, the agent is in one state (X), it finds the optimal action (U) and executes it. It then enters another state(X'). After the execution of the selected action, the agent may get a reward. The function $Q(X, U)$ is the value function for a given state(X) and action(U). It is the immediate reward

r after the execution of the action(U), plus the best Q-value (discounted by a factor γ) in the following state. The algorithm is shown in algorithm 1 [5].

Algorithm 1 Reinforcement Learning Algorithm

for each pair of $< X, U >$ **do**

$Q(X, U) \leftarrow 0$

end for

Observe the current state X

loop

1.Select an action U and execute it

2.Receive immediate reward r

3.Observe the new state X'

4.Update the table entry for Q(X,U), as follows:

$$\text{Q(X,U)} = \text{r} + \gamma * \max_{U'} Q(X', U')$$

end loop

Initially, since all function values are zero, the agent just selects an action randomly. With more and more experience, the function values may converge to the actual values [5] and the agent may use the learned function values for optimal control.

It is important to note that in general, the size of the state space determines how quickly the algorithm will converge on the correct function. The more states there are, the longer it will take to learn the correct function.

3.1 Reinforcement Learning with Continuous States

One may notice that the algorithm listed above works only for discrete states and actions. This is not the case in our path-tracking problem. Although the action of the car (i.e. left-turn, right-turn etc) is discrete, the state, a 3-tuple

vector $< \tilde{y}, \tilde{\theta}, R >$, is continuous. We must provide some sort of mechanism to quantize the state before reinforcement learning can be applied in this problem. There are at least two approaches.

The first one is to quantize the state directly and apply reinforcement learning. An example is shown in figure 2. This method is simple but inflexible and inefficient. It will unnecessarily increase the size of the state space. For example, assume that the car is facing in the right direction when following a straight line. In this case, if the car is only slightly off the line, we want to turn gently to approach the line and to not overshoot it. If the car is far away from the line, we want to turn sharply to get back onto the path. Therefore, we would require a fine quantization. But if we are following a circle, then independently of how far away we are from the outside of the circle, we want to make a sharp turn, since all circles are maximum turns. This means that the fine quantization will generate unnecessary states, which will greatly reduce the convergence speed of the algorithm.

The second method is the use of a function approximator for the value function Q. In this case, the quantization is implicit and based on previous cases, which are stored in a database. Figure 3 shows an example of a case-based quantization. A state is assigned a value based on a prototypical case (e.g., close to the straight line or further away). In this simple example, all cases have the same area of influence. The key is to calculate the distance from the current state to those existing states in the database. In the example, only the nearest case determines the type of state, but in our implementation, a nearest neighbor set is computed.

In general, this distance is used to measure the contribution of all those selected states in the database in the Q evaluation. I select the function approximator method in our implementation and it is described in more

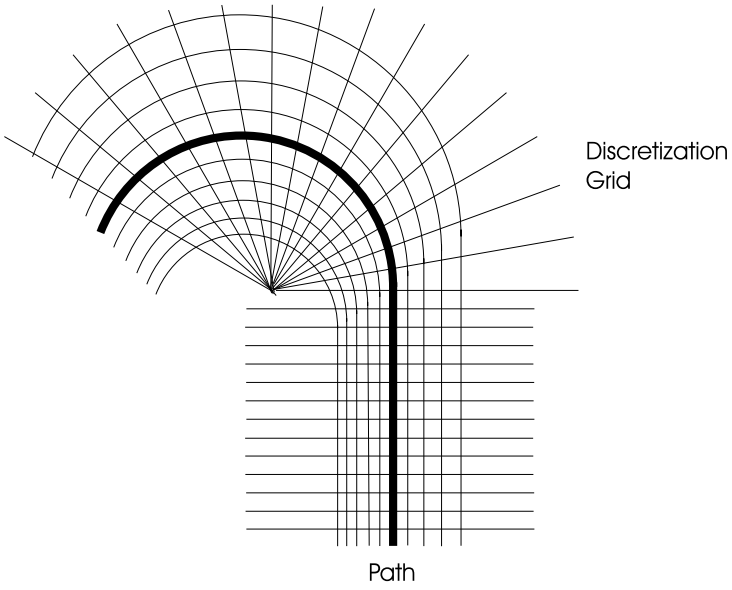

Figure 2: Static Quantization. Each intersection of lines represents of the discretization grid a world state.

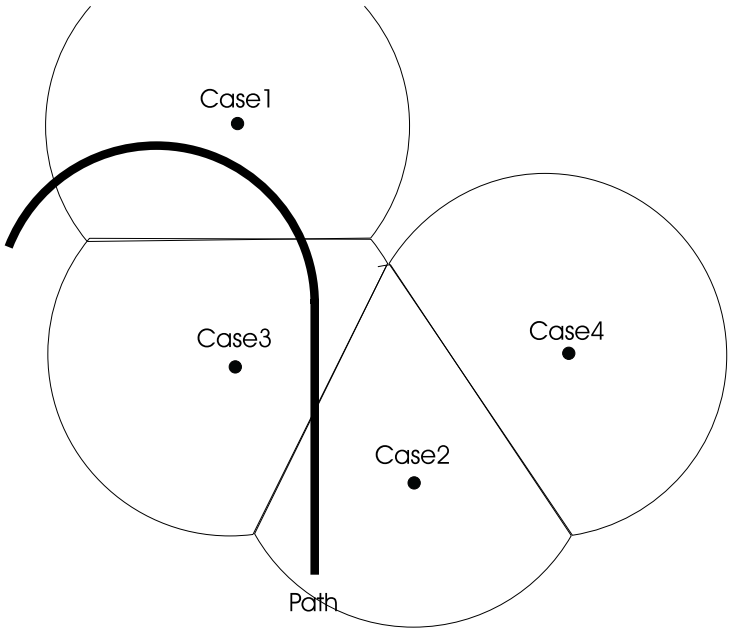

Figure 3: Case-based Quantization

detail in the following section.

4 Representation with Case-based Function Approximator

Function approximators are used to represent the value function(Q) for a continuous state problem. In discrete space, a finite resource can be used to store the value function (i.e. use a two-dimensional array for the Q value, one subscript for the state X, the other for the action U), while in continuous space this is not the case. There are many functions approximators. For more details please see [9]. The Case-based function approximator is one of them and it is suitable for our task because of its structure. Operations are defined for the evaluation and update of the value function. Details can be found in [9].

4.1 Case Structure

Every case in the database corresponds to one input point X_i that the agent has visited($X_i =< \tilde{y}_i, \tilde{\theta}_i, R_i >$ in our case). One case C_i is:

$$C_i = (X_i, Q_i, e_i, \{U_{ij}, Q_{ij}, e_{ij}\}) \tag{1}$$

where j=1... N, N is the number of actions

From 1, it can be seen that C_i consists of two separate portions, the first portion (x_i, Q_i, e_i) is associated only with the state, the other (U_{ij}, Q_{ij}, e_{ij}) is associated with actions within the state. e_i is the eligibility trace of the state [10], while e_{ij} is the eligibility trace for action j within the state i.

4.2 Function Evaluation

Whenever a query point(X_q, U_q) needs the value function for the current input X_q, the database is searched for those states that are similar to the query state. The distance (d_{iq}) from an existing state (X_i) to the query state X_q can be used for the estimation of similarity $(d_{iq} = f(\tilde{y_i} - \tilde{y_q}) + g(\tilde{\theta_i} - \tilde{\theta_q}) + h(R_i - R_q))$. After search through the entire database, a nearest neighbor set NN_q for the query state X_q is generated. NN_q consists of those states with distance to X_q less than a predefined threshold τ_k . That is

$$NN_q = \{Q_i | d_{iq} <= \tau_k\} \tag{2}$$

The distance measure d_{iq} is defined as:

$$d_{iq} = \sqrt{\left(\frac{\tilde{y_i} - \tilde{y_q}}{2}\right)^2 + \left(\frac{\tilde{\theta_i} - \tilde{\theta_q}}{2 * \pi}\right)^2 + \left(\frac{R_i - R_q}{2}\right)^2} \tag{3}$$

The distance is based on three parts: current distance error, current orientation error and the curvature of the path. The distance error is normalized to $-1..1$ meter, the orientation error to 360 degrees, and the curvature to 0..2.

From NN_q in Equation (2), all existing cases in the database that are similar to the current input X_q can be found, thus the Q value for the query point $< X_q, U_q >$ can be calculated by the following formulas:

$$Q_i(U_q) = (1-\rho)Q_i + \rho\left(\sum_{\forall u_{ij}\in C_i}\frac{K^u(d_{ij}^u)}{\sum_j K^u(d_{ij}^u)}Q_{ij}\right)\forall C_i \in NN_q \quad (4)$$

$$Q(X_q,U_q) = \sum_{\forall C_i\in NN_q}\frac{K^x(d_i^x)}{\sum_j K^x(d_j^x)}Q_i(u_q) \quad (5)$$

The action having the highest $Q(X_q,U_q)$ is used as the current action for the input X_q.

4.3 Learning Update

All Q-value in the database must be updated after a new reward is returned from the environment for the given action. The eligibility traces (e_i,e_{ij} in Equation(2)) are also updated according to their contribution to Q(X_q,U_q). Based upon the distance function, a new case is created if no case near enough to the query input X_q exists [9].

4.4 An Example of Function Evaluation

In this section, I give an example to show how to evaluate the Q-value for an input $X_q =< 0.5, 0, 1 >$ and find the best action for it. For simplicity, after searching the database, only two cases are in NN_q in Equation (2), as shown in Figure 4. Table 1 shows details of the cases in NN_q. There are only three actions(0 for left-turn, 1 for go-straight, 2 for right-turn) here. In the actual implementation I have nine different steering angles.

In Table 1 d_{iq} is calculated by Equation (3), The selection of K^u in Equation (4) and K^x in Equation (5) is based on the strategy of exploitation and exploration[5]. Let here ρ =0.6 in Equation(4), K^u be such that in

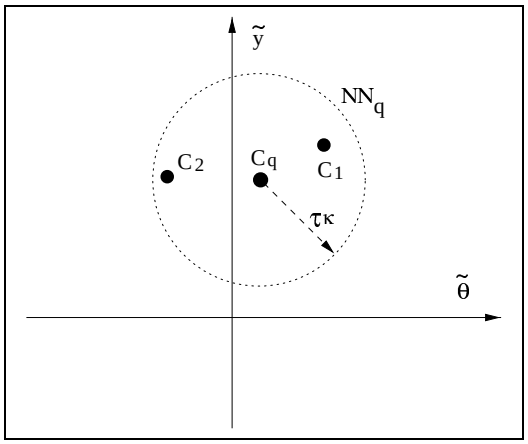

Figure 4: Two cases in the NN_q

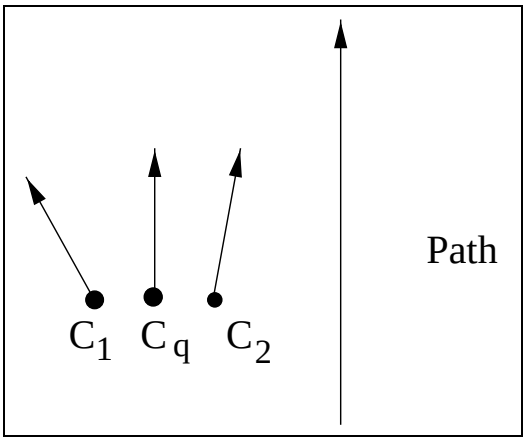

Figure 5: One of the situations as shown in the left

Case No.	$\tilde{y}$	$\tilde{\theta}$	R_i	Q	Q_0	Q_1	Q_2	d_{iq}
1	0.6	0.3	1	-0.2	-0.8	-0.6	-0.3	0.069
2	0.4	-0.1	1	-0.1	-0.7	-0.1	-0.2	0.052

Table 1: The two cases in the nearest neighbor set NN_q

Equation(4) $Q_i(U_q) = (1-\rho)Q_i + \rho Q_{iq}$ (that is, only the value of the action that is the same as the query action is considered), $K^x(d_i^x) = d_i^x$.

$$d_{1q} = \sqrt{\left(\frac{0.6-0.5}{2}\right)^2 + \left(\frac{0.3-0}{2\pi}\right)^2 + \left(\frac{1-1}{2}\right)^2} = 0.069$$
$$d_{2q} = \sqrt{\left(\frac{0.4-0.5}{2}\right)^2 + \left(\frac{-0.1-0}{2\pi}\right)^2 + \left(\frac{1-1}{2}\right)^2} = 0.052$$

$$Q_1(0) = (1-\rho)Q_1 + \rho Q_0 = (1-0.6)*(-0.2) + 0.6*(-0.8) = -0.56$$
$$Q_1(1) = (1-\rho)Q_1 + \rho Q_1 = (1-0.6)*(-0.2) + 0.6*(-0.6) = -0.44$$
$$Q_1(2) = (1-\rho)Q_1 + \rho Q_2 = (1-0.6)*(-0.2) + 0.6*(-0.3) = -0.26$$
$$Q_2(0) = (1-\rho)Q_2 + \rho Q_0 = (1-0.6)*(-0.1) + 0.6*(-0.7) = -0.46$$
$$Q_2(1) = (1-\rho)Q_2 + \rho Q_1 = (1-0.6)*(-0.1) + 0.6*(-0.1) = -0.1$$
$$Q_2(2) = (1-\rho)Q_2 + \rho Q_2 = (1-0.6)*(-0.1) + 0.6*(-0.2) = -0.16$$
$$Q(X_q,0) = \frac{d_{1q}}{d_{1q}+d_{2q}}Q_1(0) + \frac{d_{2q}}{d_{1q}+d_{2q}}Q_2(0) = \frac{0.069}{0.121}(-0.56) + \frac{0.052}{0.121}(-0.46) = -0.52$$
$$Q(X_q,1) = \frac{d_{1q}}{d_{1q}+d_{2q}}Q_1(1) + \frac{d_{2q}}{d_{1q}+d_{2q}}Q_2(1) = \frac{0.069}{0.121}(-0.44) + \frac{0.052}{0.121}(-0.1) = -0.29$$
$$Q(X_q,2) = \frac{d_{1q}}{d_{1q}+d_{2q}}Q_1(2) + \frac{d_{2q}}{d_{1q}+d_{2q}}Q_2(2) = \frac{0.069}{0.121}(-0.26) + \frac{0.052}{0.121}(-0.16) = -0.22$$

As the $Q(X_q, 2)$ has the highest value, the agent will take action 2, namely turn right when the input X_q is $< 0.5, 0, 1 >$

5 Experiments

The controller described above has been implemented both in simulation and practical driving. Surprisingly, the database generated during simulation can be directly applied to practical driving. This means that the controller in a real world environment does not need to learn from scratch, which is very difficult in practice because of it requires too many training episodes and because you need to put the car close to the path again if the current trial fails.)

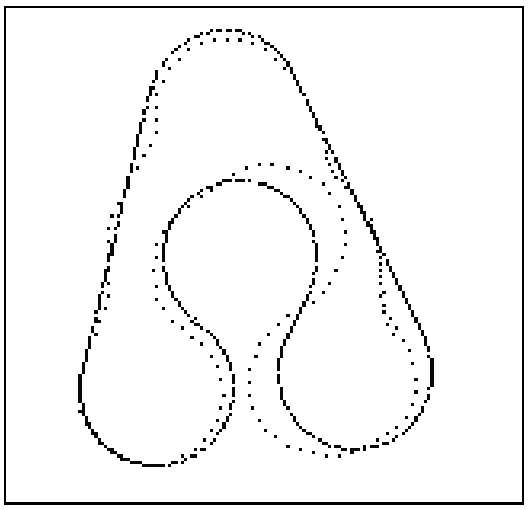

Figure 6: Learned result in simulation after 1000 trials

The Aucklandianapolis race track is used as the sample path, both in simulation and practical driving.

Table 2 shows the average position and orientation errors for different numbers of learning episodes. Each trial consists of 200 steps. The data is averaged in 100 trials after trials shown in the column 'Trials' are done from scratch.

In our practical driving, the learned result can be used to control the toy car. Though the improvement on performance through learning is quite obvious in simulation, it's not the case in practice. Work continues to improve the learning when driving the real car.

6 Conclusions

Reinforcement learning can be adapted to control a car in path-tracking. In this paper we describe some aspects of reinforcement learning, such as

Experiment	Trials	Average $\tilde{y}$(m)	Average $\tilde{\theta}$(radius)
1	200	0.2684	0.3202
2	400	0.2126	0.2802
3	600	0.0734	0.1381
4	800	0.0462	0.1043
5	1000	0.0509	0.1033
6	2000	0.0477	0.0943

Table 2: Average Control Errors in Simulation

function value representation and how it is used in the problem of path-tracking. The performance (average control errors) of the simulation in our experiment is satisfactory. The learned values in the simulation can also be used in the real world task of driving our toy cars. However, further work needs to be done to achieve significant improvement as that in the simulation in the real world with its many more sources of errors such as noise, actuator error and slipping.

References

[1] A. Balluchi, A. Bicchi, A. Balestrino, and G. Casalino. Path tracking control for dubin's cars. In *Proceedings of IEEE International Conference on Robotics and Automation*, Centro "E. Piaggio" & Dipartimento Sistemi Elettrici e Atuomazione, Universitá di Pisa., 1996.

[2] Jacky Baltes. Aucklandianapolis homepage. WWW, February 1998. http://www.tcs.auckland.ac.nz/ jacky/teaching/courses/-415.703/aucklandianapolis/index.html.

[3] Antonio Bicchi, Giuseppe Casalino, and Corrado Santilli. Planning shortest bounded-curvature paths for a class of nonholomic vehicles among obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1349–1354, 1995.

[4] Alessandro De Luca, Giuseppe Oriolo, and Claude Samson. Feedback control of a nonholonomic car-like robot. Technical report, Universita di Roma La Sapienza, 1996.

[5] Tom Mitchell. *Machine Learning.* mcGraw-Hill, 1997.

[6] Ben Noonan, J. Baltes, and B. MacDonald. Pc interface for a remote controlled car. *In Proc. Of the IPENZ sustainable city conference*, pages 22–27, 1998.

[7] Robin Otte. Path following control of a nonholonomic vehicle at high speeds using a fuzzy controller. Technical report, Computer Science, The University of Auckland, rott001@cs.auckland.ac.nz, 1998.

[8] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter 20, pages 598–624. Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632, 1995.

[9] JC Santamaria, RS Sutton, and A Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6:163–217, 1997.

[10] Singh SP and Sutton RS. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 1996.