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

Jacky Baltes and Yuming Lin

Centre for Image Technology and Robotics
University of Auckland,Auckland
New Zealand
j.baltes@auckland.ac.nz
http://www.citr.auckland.ac.nz/~jacky

Abstract. This paper investigates the use of reinforcement learning in solving the path-tracking problem for car-like robots. The reinforcement learner uses a case-based function approximator, to extend the standard reinforcement learning paradigm to handle continuous states. The learned controller performs comparable to the best traditional control functions in both simulation and also in practical driving.

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 controls. A parallel port micro-controller based interface ([7]) 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. The toy cars do not have shaft encoders so there is a feed forward control error when driving a given path.

Therefore, we need a controller which keeps the car on the track. Note that the control function described in this paper only depends on the curvature of the path and is thus mostly independent of the path itself. This means that our results are also applicable to more dynamic environments, such as RoboCup.

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 [5].
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 practice. The

motivation of this project was to improve on its performance in the real world, see section (5).
3. A Fuzzy logic controller [8] 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. At the core of the reinforcement learner is a value function, called Q-value, which is why it is also called Q-Learning ([9]).

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 to approximate 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 Fig. 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, $\tilde{\theta} = \theta - \hat{\theta}$).

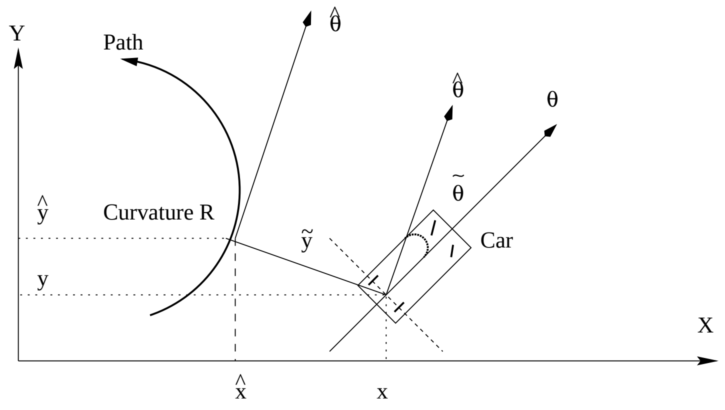

Fig. 1. The Kinematic Model

In the representation used in this paper, the current state of the system is defined by the positional error $\tilde{y}$, the orientation error $\tilde{\theta}$ 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 outputs are desired settings for speed and direction.

3 Reinforcement Learning

This section gives a brief introduction to reinforcement learning. The most important concepts in reinforcement learning are the agent and the environment. 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 from the environment about how good or bad a certain state is. This feedback is called *reward.* The task of reinforcement learning is to find the action with the highest expected reward (*Q-value*) in the current state. In the path tracking problem, 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. Usually, the selected action is the one with the highest expected reward (*Q-value*). To prevent premature convergence on suboptimal action sequences, a reinforcement learner will sometimes not select the best action so that it can further explore the environment. This is called the exploration vs. exploitation trade-off.

After executing the action, the agent enters another state(X'). The agent may get a reward (positive or negative reinforcement) when entering certain states. 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 reinforcement learning algorithm is shown below (Algorithm 1).

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:
            Q(X,U) = 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 [6] 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 larger the state space, the longer it will take to learn the correct function.

3.1 Reward function in the car domain

The reward function is of critical importance in the design of a reinforcement learner. The reward function must accurately reflect the progress that an agent is making towards achieving a goal, since otherwise the agent will learn the wrong behavior.

In the car driving domain, we want to keep the car on the path, so it is reasonable to base the reward function on the position ($\tilde{y}$) and orientation error ($\tilde{\theta}$). Preliminary experiments, however, with Balluchi's controller ([1]) suggested that it is also important to have a smooth control function. Therefore, the reward function in this research is based on the weighted sum of normalized position error ($\tilde{y}$) and orientation error ($\tilde{\theta}$) as well as the necessary control work (Difference in control setting U at time t and time $t-1$) as shown in Equation 1.

$$r = -w_1 * (\frac{\tilde{y}}{2})^2 + w_2 * (\frac{\tilde{\theta}}{2\pi})^2 + w_3 * (\frac{U_t - U_{t-1}}{9})^2 \tag{1}$$

In a control problem, in principle a reward can be associated with every state. However, to get a better estimate of the real reward of a state, we return as reward the sum of the rewards for the last five states.

3.2 Reinforcement Learning with Continuous States

One may notice that the algorithm listed above assumes discrete states and actions. This is a problem in our path-tracking domain. Although the actions of the car (i.e. left-turn, right-turn etc) are discrete, the state, a 3-tuple vector $< \tilde{y}, \tilde{\theta}, R >$, is continuous. We must provide some mechanism to quantize the state space 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 to the right of the line, we want to turn gently left 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.

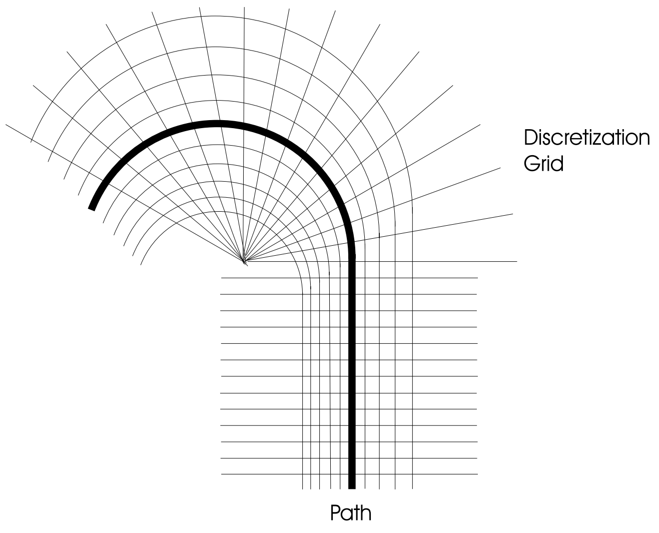

Fig. 2. Static Quantization. Each intersection of lines represents of the discretization grid a world state.

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. The research described in this paper uses a case-based function approximator, which is described in more detail in the following section.

4 Reinforcement learning using a 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, whereas in continuous space this is not the case. There are many functions approximators. For more details please see [10]. 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 [10].

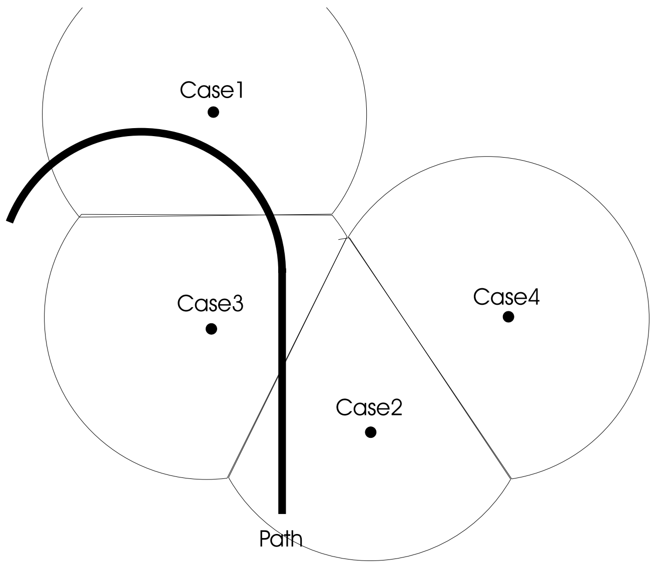

Fig. 3. Case-based Quantization

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:

$$\begin{array}{l} C_i = (X_i, Q_i, e_i, \{U_{ij}, Q_{ij}, e_{ij}\}) \\ \text{where } i = 1 \ldots N \text{ is the number of cases} \\ \text{and } j = 1 \ldots M\text{, M is the number of actions} \end{array} \tag{2}$$

From 2, 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 [11], while e_{ij} is the eligibility trace for action j within the state i.

4.2 Function Evaluation

To evaluate the value function $Q(X_Q, U_Q)$ for the query state X_Q, the database is searched for those states that are similar to the state in question. The distance (d_i) from an existing state (X_i) to the query state X_q can be used for the estimation of similarity $(d_i = f(\tilde{y_i} - \tilde{y_q}) + g(\tilde{\theta}_i - \tilde{\theta}_q) + h(R_i - R_q))$. 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

$$NN_q = \{Q_i | d_i <= \tau_k\} \tag{3}$$

The distance measure d_i is defined as:

$$d_i = \sqrt{\left(\frac{\tilde{y_i} - \tilde{y_q}}{2}\right)^2 + \left(\frac{\tilde{\theta_i} - \tilde{\theta_q}}{2 * \pi}\right)^2 + \left(\frac{R_i - R_q}{2}\right)^2} \tag{4}$$

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 (3), 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:

$$Q_i(U_q) = (1-\rho)Q_i + \tag{5}$$
$$\rho\left(\sum_{\forall u_{ij} \in C_i} \frac{K^u(d^u_{ij})}{\sum_j K^u(d^u_{ij})} Q_{ij}\right) \forall C_i \in NN_q$$

$$Q(X_q, U_q) = \sum_{\forall C_i \in NN_q} \frac{K^x(d^x_i)}{\sum_j K^x(d^x_j)} Q_i(U_q) \tag{6}$$

The value $Q_i(U_q)$ is the overall Q value for the current query action U_q in state X_i. It consists of two parts: (a) the Q value for state X_i and (b) the sum over all actions in this state.

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

4.3 Learning Update

All Q-values 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(3)) 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 [10].

4.4 An Example of Function Evaluation

This section gives an example of how to evaluate the Q-value for an input $X_q =< 0.5, 0, 1 >$ and to find the best action for state X_Q. For simplicity, after searching the database, only two cases are in NN_q in Equation (3), 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. The actual implementation uses nine different steering angles.

In Table 1, d_{iq} is calculated by Equation (4), The selection of K^u in Equation (5) and K^x in Equation (6) is based on the strategy of exploitation and exploration[6]. Set $\rho = 0.6$ in Equation(5), and let K^u be such that in Equation(5)

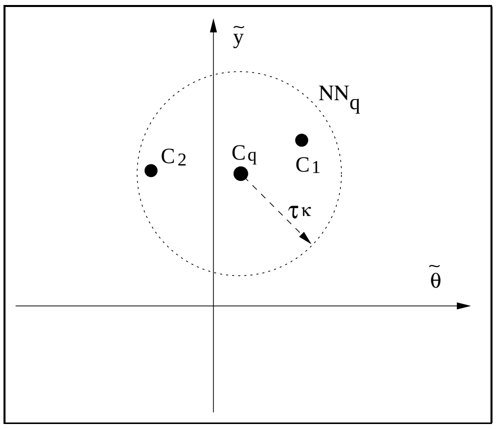

Fig. 4. Two cases in the NN_q

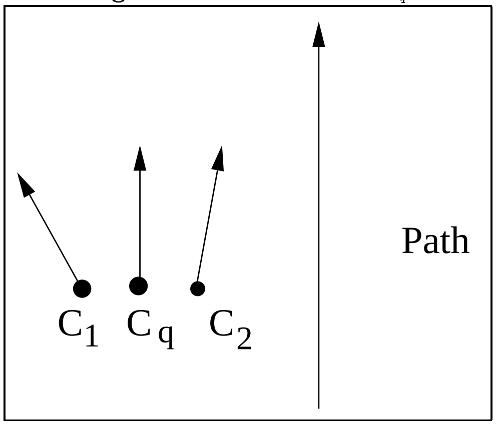

Fig. 5. One of the situations as shown in Fig. 4

Case	$\tilde{y}$	$\tilde{\theta}$	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

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

The distance of case 1 and 2 to the query state are given as:

$$d_1 = \sqrt{\left(\frac{0.6-0.5}{2}\right)^2 + \left(\frac{0.3-0}{2\pi}\right)^2 + \left(\frac{1-1}{2}\right)^2} = 0.069$$

$$d_2 = \sqrt{\left(\frac{0.4-0.5}{2}\right)^2 + \left(\frac{-0.1-0}{2\pi}\right)^2 + \left(\frac{1-1}{2}\right)^2} = 0.052$$

The overall Q values for the actions in state X_1 and X_2 are computed using Equation 5.
$Q_1(0) = (1-\rho)Q_1 + \rho Q_0 = (1-0.6)*(-0.2) + 0.6*(-0.8) = -0.56$
$Q_1(1) = (1-\rho)Q_1 + \rho Q_1 = (1-0.6)*(-0.2) + 0.6*(-0.6) = -0.44$
$Q_1(2) = (1-\rho)Q_1 + \rho Q_2 = (1-0.6)*(-0.2) + 0.6*(-0.3) = -0.26$

So action 2 (right turn) has the best Q value in case 1.
$Q_2(0) = (1-\rho)Q_2 + \rho Q_0 = (1-0.6)*(-0.1) + 0.6*(-0.7) = -0.46$
$Q_2(1) = (1-\rho)Q_2 + \rho Q_1 = (1-0.6)*(-0.1) + 0.6*(-0.1) = -0.1$
$Q_2(2) = (1-\rho)Q_2 + \rho Q_2 = (1-0.6)*(-0.1) + 0.6*(-0.2) = -0.16$

Similarly, action 1 (straight) has the best Q value for case 2. As shown below, we evaluate the best action for the current state X_q by using Equation 6.

$Q(X_q, 0) = \frac{d_1}{d_1+d_2}Q_1(0) + \frac{d_2}{d_1+d_2}Q_2(0) = \frac{0.069}{0.121}(-0.56) + \frac{0.052}{0.121}(-0.46) = -0.52$

$Q(X_q, 1) = \frac{d_1}{d_1+d_2}Q_1(1) + \frac{d_2}{d_1+d_2}Q_2(1) = \frac{0.069}{0.121}(-0.44) + \frac{0.052}{0.121}(-0.1) = -0.29$

$Q(X_q, 2) = \frac{d_1}{d_1+d_2}Q_1(2) + \frac{d_2}{d_1+d_2}Q_2(2) = \frac{0.069}{0.121}(-0.26) + \frac{0.052}{0.121}(-0.16) = -0.22$

As $Q(X_q, 2)$ 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 it requires too many training episodes and because you need to put the car close to the path again if the current trial fails.)

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 over 100 trials after the training phase.

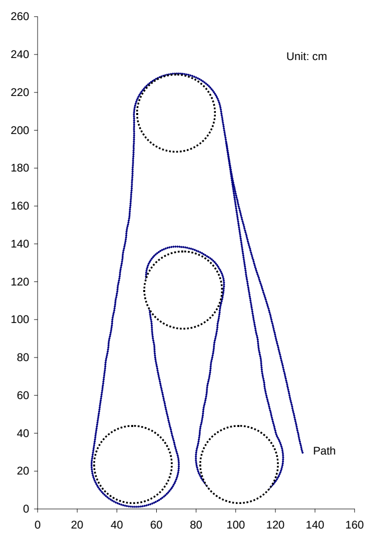

Fig. 6. Learned result in simulation after 1000 trials

Figure 7 shows the result of using the learned controller to drive the car in practice. As can be seen, the controller can use the results from simulation to *seed* the controller to drive the car in practice. Ideally, one would like the controller to improve its performance now in practice with increased experience. However, there is no noticeable improvement in practical driving. There are two reasons: (a) the reinforcement learner has settled, that is most Q values are known, on the current controller so that it is unlikely that it will explore new actions, and (b) it takes a lot longer to drive a track in practice as opposed to simulation where it is easy to drive a few thousand laps.

Experiment	Training	Avg. $\tilde{y}$(m)	Avg. $\tilde{\theta}$(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

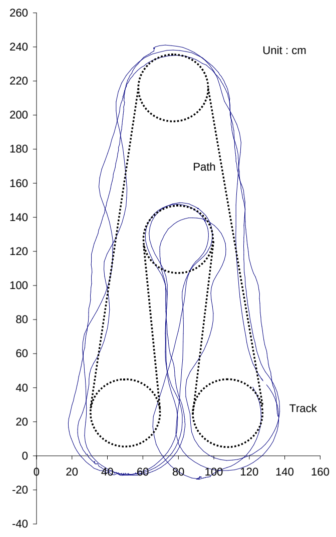

Fig. 7. Using the learned result in practical driving

6 Conclusions

In this paper we describe some aspects of reinforcement learning, such as function value representation and how it is used in the problem of path-tracking. Reinforcement learning can be adapted to control a car in path-tracking. The training can be initialized using a simulation and after the performance has stabilized, training can continue with practical driving. Since the representation is independent of the path ahead, the learned controller can be used in real time path following tasks, independently of whether the path is static (as in the Aucklandianapolis) or dynamic (as in the RoboCup competition).

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.

The reinforcement learning controller has also proven itself in the Aucklandianapolis competition. It won the 1999 competition in 2 minutes 30 seconds, which is twice as fast as Balluchi's controller and comparable to the Fuzzy Logic controller ([8]). The reinforcement controller is the only controller that has been used successfully to drive cars with and without linear steering behavior. The Fuzzy Logic controller works by interpolating steering angles and thus works only for cars with at least an approximate steering behavior.

However, further work is needed to achieve significant improvement during learning (as in the simulation) in the real world with its many sources of errors e.g., noise, actuator error and slipping.

Another improvement is the use of dynamic weights in the reward function. A simple example shows that given our representation, static weights are insufficient to learn the correct control function. Assume that the car is following a straight line. When the car is far away from the path, then it will need to steer towards the path, which means that it will get a reduced reward due to the orientation error. In this case, the weight on the orientation error should be small. However, when close to the line, the orientation error is more important than the position error and should receive more weight. Full details of the practical evaluation of the reinforcement learner are given in [4].

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, Universită 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. Yuming Lin. *Path Tracking Control of Non-Holonomic Car-Like Robots with Reinforcement Learning.* Master's thesis, University of Auckland, Auckland, New Zealand, July 1999.
5. Alessandro De Luca, Giuseppe Oriolo, and Claude Samson. Feedback control of a nonholonomic car-like robot. Technical report, Universita di Roma La Sapienza, 1996.
6. Tom Mitchell. *Machine Learning.* mcGraw-Hill, 1997.
7. 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.
8. 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.
9. Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter 20, pages 598–624. Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632, 1995.
10. 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.
11. Singh SP and Sutton RS. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 1996.