Proceedings of the 2006 IEEE International Conference on Robotics and Automation
Orlando, Florida - May 2006

On-line Optimal Motion Planning for Nonholonomic Mobile Robots

Tomás Martínez-Marín
Dept. of Physics, Systems Engineering and Signal Theory
University of Alicante
Alicante, Spain
E-mail: tomas@dfists.ua.es

***Abstract*— In this paper we propose a novel approach for on-line motion planning of nonholonomic robots through reinforcement learning. The algorithm incorporates a mechanism, the adjoining property, to select the state transitions that will be learned by the robot controller. The method overcomes some limitations of reinforcement learning techniques when they are employed in applications with continuous non-linear systems, such as nonholonomic vehicles. Furthermore, a good approximation to the optimal behaviour is obtained by a look-up table without of using function interpolation. Finally, we present both simulation and experimental results to show the satisfactory performance of the method compared with the popular Q-learning algorithm.**

***Index Terms*— Optimal motion planning, Reinforcement learning, Cell-to-cell mapping.**

I. INTRODUCTION

Optimal motion planning of nonholonomic vehicles is today an open problem especially in complex tasks (e. g., parallel parking, stop-and-go mode in traffic jams). Car-like vehicles are widely used in industry since they have the necessary loading capability with only a power motor. These vehicles are nonlinear dynamic systems, whose motion laws have been a well studied topic [1-3]. However, the motion planning and control of car-like robots are difficult tasks since they are nonholonomic systems. For this reason, the problem has usually been decomposed in two steps: path planning and motion control. Path planning consists of computing a collision-free motion using a map of the environment while the motion control concerns the execution of the motion. Our approach addresses the optimisation of the first step, since intelligent vehicles should exhibit an optimal behaviour in real scenarios. This supposes an additional aim in the design of efficient algorithms for motion planning in autonomous robots with restricted computational resources.

A number of planning algorithms have been proposed for generating trajectories that give rise to smooth motion [4-5]. These trajectories are computed in open-loop, but in real environments where a vehicle is subject to perturbations and uncertainty, closed-loop action is more desirable.

Minimal length paths of simplified car-like vehicles have been characterized in [2]. This result is proven in the absence of obstacles and it is computed in open-loop. Dynamic Programming (DP) provides a closed-loop solution including obstacles [6]. Although DP is efficient compared with direct search, it requires a lot of computational resources and a model of the robot motion law. In this paper, we propose a new reinforcement learning algorithm to solve the optimal path planning problem in an efficient manner compared with others methods such as Q-learning. The algorithm is an extension of the CACM technique [7], which is based on cell-to-cell mapping techniques and Bellman's principle of optimality for continuous dynamic systems.

Reinforcement learning (RL) methods provide a great advantage with respect to the cited approaches. The optimal motion law of a mobile robot is estimated on-line while the robot is interacting with the environment. Thus, the RL controller does not require a previous vehicle model to obtain an optimal behaviour of the real vehicle. In this context, there are several applications of robot motion planning by reinforcement learning. For example, in [8] a robot docking task was solved through RL in a visual servoing framework. The optimal motion of a car-like robot is slightly more difficult since the system is nonholonomic and the dimension of the problem is higher (3D or more).

The paper is organized as follows. In section 2 we present some basic concepts of reinforcement learning. Section 3 provides a brief introduction to the Cell Mapping techniques. Section 4 describes the vehicle platform. Implementation aspects of the new algorithm are addressed in section 5, tested in section 6 and 7 through simulations and experimentation on a real mobile robot, respectively. Conclusions appear in section 8.

II. REINFORCEMENT LEARNING

Reinforcement learning methods only require a scalar reward (or punishment) to learn to map situations (states) in actions [9]. As opposed to supervised learning, they do not require a teacher to acquire the optimal behaviour, they

0-7803-9505-0/06/$20.00 ©2006 IEEE

only need to interact with the environment learning from experience. The knowledge is saved in a look-up table that contains an estimation of the accumulated reward to reach the goal from each state. The objective is to find the actions (policy) that maximize the accumulated reward in each state.

Q-learning is one of the most popular reinforcement learning methods, since with a simple formulation it can address model-free optimization problems. The accumulated reward for each pair state-action *Q(s,a)* is updated (back-up) by the one-step equation

$$\Delta Q(s,a) = \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \quad (1)$$

where *Q(s,a)* is the expected value of performing action *a* in state *s*, *r* is the reward, *s'* is the resultant state after a transition, 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. If the reward function is *proper* [10] the discount factor can be omitted ($\gamma = 1$). The action *a* with highest *Q* value at state *s* is the best policy up to instant *t*.

In applications with real systems Q-learning spends long time making hundred of thousands of trials to approximate the optimal behaviour. To speed up the learning process it is necessary to incorporate some planning mechanism. Prioritized sweeping [11] and Dyna-Q include a search control to simulate the past real experiences in a specified order. In our simulations we will combine Q-learning with the search mechanism proposed in the new RL algorithm. Thus, we will compare the quality of the optimal solution more than the learning rate to acquire the optimal behaviour.

III. Cell Mapping Techniques

Cell-to-cell mapping methods are based on a discretization of the state variables of the system, defining a partition of the state space into cells [12]. A cell-to-cell mapping can be derived from the dynamic evolution of the system. In [13] a cell mapping application for the design of optimal controllers is proposed. This method (CSCM), based on the Simple Cell Mapping (SCM) technique, carries out a discretization of both state and control variables and uses a cost function to specify the desired optimality criterion. In [7] the CACM algorithm for optimal control of highly nonlinear systems is proposed. This method is based on the Adjoining Cell Mapping (ACM) technique, whose central concept is the creation of a cell mapping where only transitions between adjoining cells are allowed [14].

The adjoining property states that the distance D_k between the current cell and the previous cell is equal to some integer value *k* equal or greater than 1. For our RL controller, we will define the adjoining property in terms of the continuous states **x** and **x'** as follows:

$$D_k(\mathbf{x},\mathbf{x}') = \max_j \left| \frac{x_j - x'_j}{h_j} \right| = k, \quad (2)$$

where x_j indicates the *j*-th component of the state **x** and h_j is the cell size of the *j*-th dimension.

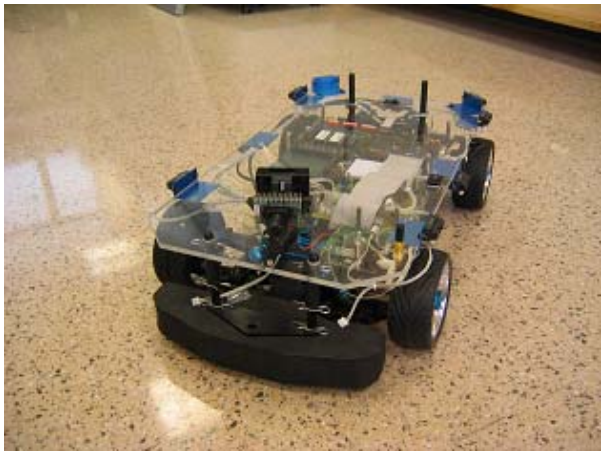

Fig. 1. The autonomous vehicle employed for the experiments.

In Q-learning the transitions between states are evaluated at fixed sample times, while with our RL controller the transitions have to satisfy the adjoining distance condition in order to be evaluated. Selecting appropriately this distance with respect to the number of cells of the system it is possible to minimize the effect of the grid resolution and the relative length of the transitions over the optimal behaviour of the system.

Bellman's Principle of Optimality states a strategy to determine the global optimal solution for dynamic systems. The solution can be obtained through the well-known Bellman equation, or by addressing the problem as a shortest path search, since shortest path problems satisfy the Principle of Optimality. Both methods find the optimal policy by processing the state space backward in time, starting at the objective state. The CACM technique carries out the shortest path search in an efficient manner, reducing both memory requirements and computation time. For details of the algorithm see [7].

In continuous state spaces the CACM method is an attractive alternative to conventional dynamic programming since it states an efficient method for managing the approximation to the optimal behaviour without incurring a high computational cost. Some limitations such as chattering effects in the control variables can be removed by linear interpolation or other methods [15].

IV. Vehicle Platform

The reinforcement learning controllers have been implemented in the nonholonomic mobile robot shown in Figure 1. The vehicle is equipped with an array of infrared sensors, sonar and a compass. The robot is autonomous,

using a microcontroller MPC555 to process all sensors and the RL controllers.

In order to simulate the car-like motion we will employ a simple model, although this model will not be used by the algorithm in order to find the optimal system behaviour. According to its locomotion system, we can model a car-like vehicle through state space formulation [2] as

$$\begin{aligned} \dot{x} &= v\cos(\theta)\cos(\varsigma), \\ \dot{y} &= v\sin(\theta)\cos(\varsigma), \\ \dot{\theta} &= v\sin(\varsigma). \end{aligned} \tag{3}$$

The distance between the reference point (x, y) and the middle point of the driving wheels is assumed to be 1. The orientation of the car is denoted by θ. The two controls of a car are the velocity v of the driving wheels and the steering angle ζ. It is important to note that the state equations are only used in the simulations in order to describe the robot trajectories. In the experiments, the model is built on-line by recording the transitions between states and the immediate rewards.

Although this simple model has its limitations, and more complex models can be found, for example, in [4], for the purpose of this paper it should suffice. The application of the RL technique considering more state variables only affects the computational resources, but does not increment the problem complexity.

Observing the state space equations, we can see that the time derivatives of the state variables only depend on the orientation variable. Therefore, to obtain information about the relative motion of the car (i.e., transitions between adjoining cells) it is enough to obtain the cell transitions only for the third state variable, starting each transition from the origin in the xy plane. The former is only valid if the low level controllers can keep the specified velocities on different surfaces. Using a transformation matrix (T) we can exploit this property. Thus, after learning a few local trajectories, the algorithm only spends time searching for the global optimal policy.

V. The Algorithm

Q-learning was conceived for discrete state and action spaces. In robot motion planning applications the state space is continuous (infinitum states), so it is mandatory to discretize the state space in order to use Q-learning as it was conceived. The state space is divided in cells, containing each cell infinitum states that will be represented for the algorithm as a single state. The inherent discretization errors can produce a poor approximation to the optimal behaviour in complex nonlinear systems such as car-like vehicles.

The CACM algorithm incorporates the adjoining distance mechanism to deal with the discretization problem. In Q-learning the transitions between states are evaluated at fixed sample times, while in CACM the transitions have to satisfy the adjoining distance condition in order to be evaluated.

A. Description

The new algorithm has been implemented as a model-based reinforcement learning method, where the back-ups are made by simulation following the shortest path search backward in time, starting from the goal state. In direct reinforcement learning, the back-ups are only made by experimentation, which is suitable when the back-up time for experimentation compared with simulation is not very high. In general, model-based reinforcement learning methods find better trajectories and are more efficient than direct reinforcement learning.

The RL algorithm deals with several data structures for organizing the available information and storing the partial results of the learning process. These structures are the following:

- *Q(s,a)*: is the Q-value table where the accumulated reward for the (*s,a*)-pair is saved. From this table the optimal policy is obtained as follows:
$$a^*(s) = \arg\max_a Q(s,a)$$
- *M(k, x, a)*: is the local vehicle model. It contains an average of relative transitions of the car transformed to local coordinates during the learning process. The transitions have to satisfy the D-k adjoining property to assure a good approximation to the optimal policy.
- *Queue*: contains the states to be updated in the correct order to find the optimal policy in only one sweeping.
- *policy(s, M, Q)*: selects a ε-greedy policy to estimate the model (*M*) and to exploit the best policy acquired (*Q*).
- *T*: is an operator that transforms the state *s* in global coordinates to the state *x* in local coordinates.
- *distance(s, s')*: is the maximum norm between states *s* and *s'*.
- α_Q, α_M: are the learning rates for real experiences and simulated backups respectively. To prioritize the real experiences with respect to planning α_Q should be greater than α_M.

B. General procedure

The algorithm for non-linear continuous systems and deterministic environments is listed below (see [9] for notation details):

Initialize *Q(s,a), M(k,x,a)* and *first(Queue)* ← *goal*

Do forever:

1. *s* ← current state
2. *a* ← *policy(s,M,Q)*
3. Execute action *a*; observe resultant state *s'* and *r*

4. If *D-k* adjoining, repeat for $k = 1, 2, .., K$:
   $M(k,x',a) \leftarrow x = T^{-1}(s_k)$, r_k
5. $k = min\{K, distance(goal,s)\}$
6. $\Delta Q(s,a) = \alpha_Q[r_k + \gamma max_{a'} Q(s_k',a') - Q(s,a)]$
7. Repeat *N* times, while *Queue* is not empty:
   $s_k' \leftarrow first(Queue)$
   For all (s, a) that lead to s_k':
   $s = T(x)$, $r_k \leftarrow M(k,x',a) \leftarrow x' = T^{-1}(s_k')$
   $\Delta Q(s,a) = \alpha_M[r_k + \gamma\ max_{a'}\ Q(s_k',a') - Q(s,a)]$
   Insert *s* into *Queue*, sorting by $max_a Q(s,a)$
8. If *Queue* is empty, then $first(Queue) \leftarrow goal$

The algorithm previously described finds the optimal system behaviour in a highly efficient way, since it only processes and stores the necessary transitions to obtain a good approximation to the optimal motion law in the system controllable region.

VI. Optimal Motion in the Presence of Obstacles

In this section we consider the optimal motion planning in the presence of obstacles, where we study the effect of the relative length of the transitions over the optimal behaviour of the system. First, two performance indexes are defined in order to evaluate the path planners proposed.

A. Performance indexes

To demonstrate the applicability of the new RL algorithm, we measure the *percentage of controllable cells* and the *average optimal time*. A cell is controllable if starting at the center point of such a cell (initial state), the system evolves reaching some point inside the objective cell (final state). Thus, the percentage of controllable cells is the number of controllable cells with respect to the number of cells inside the region of interest that do not belong to obstacles. With a dynamic systems approach, the objective cell represents the equilibrium point of the system. The size of the corresponding domain of attraction is the number of controllable cells. On the other hand, the average optimal time indicates the approximation to the optimal behavior of the system. Both performance indexes can provide a robust test of performance of nonlinear dynamic systems such as autonomous vehicles, instead of testing the performance just for a few initial states.

B. Optimal motion planning in the presence of obstacles

To test the performance of the proposed method, we have considered moving the vehicle in an area with many obstacles, such as a parking zone. In this example, the aim of the RL controller is to move the vehicle from any initial position inside the region of interest (21×21 square meters) to a final position $P_f = (0, -9, 0)$, through a minimum-time trajectory, considering the obstacles constraints.

The entry data provided to the motion planning program are listed below:

- State variables: 3.
  - x_1: $-10.5 \le x \le 10.5$ m. Cells: 41
  - x_2: $-10.5 \le y \le 10.5$ m. Cells: 41
  - x_3: $-\pi \le \theta \le \pi$ rad. Cells: 41
- Control variables: 2.
  - u_1: $-\pi/5 \le \zeta \le \pi/5$ rad. Levels: 11
  - u_2: $-1.0 \le v \le 1.3$ m/s. Levels: 2
- Sampling time: 0.2 seconds.

TABLE I
Results of Average Optimal Time and Controllable Cells

Method	D-k/Ts	Average optimal time (s)	Controllable cells (%)
RL	D-1	25.86	73.54
	D-2	20.09	99.31
	D-3	23.78	99.61
Q-learning	0.4	33.67	54.24
	0.6	25.88	71.33
	0.8	20.24	88.04
	0.9	21.78	97.07
	1.0	22.01	90.77
	1.2	12.80	0.12

The reward function is:

$$r = \begin{cases} 100 & if\ s' = goal \\ -20 & if\ obstacle\ or\ sink \\ -\left(t + w/d^2\right) & otherwise \end{cases}$$

where t is the transition time, d is the distance (in cells) between the state s and the closest obstacle or sink, and w is a weight to balance the time optimal behaviour with the obstacle avoidance. In the simulations w was fixed to 5. Besides of avoiding obstacles, the robot is also punished with a reward $r = -20$ to prevent hitting them. The robot only gets a positive reward $r = 100$ when it reaches the goal.

The obstacles in the cellular state space are labelled as sink cells. Moreover, it is advisable to define a safety area, with the purpose of avoiding eventual collisions against obstacles. In many cases, the optimal trajectory is the nearest one that avoids the obstacle (e.g., minimum time problems); therefore the collision occurs when the trajectory suffers a deviation with respect to the ideal trajectory.

The results of a set of simulations are depicted in Table 1. The learning time was 100 seconds for all simulations. This short learning time was enough to obtain a very good approximation of the optimal motion behaviour for the RL algorithm. The high number of controllable cells demonstrates the good performance of the proposed method (e.g., the simulation employing RL D-2 with 41 cells/axis reaches a 99.31 percent of controllable cells with a low average optimal time).

We can also appreciate that the performance of the Q-learning method strongly depends on the transition time. If the transition time is not adequate, the final behaviour can be very poor.

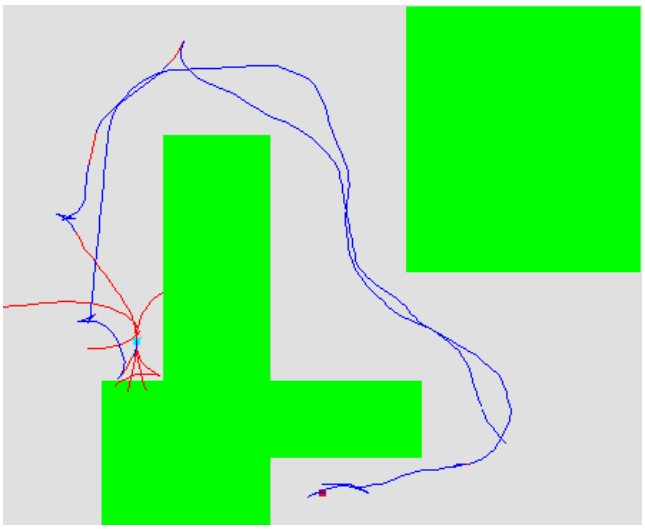

Fig. 2. Trajectories of the vehicle during the learning process. The car combines exploration (trajectories in red/grey) and exploitation (trajectories in blue/black).

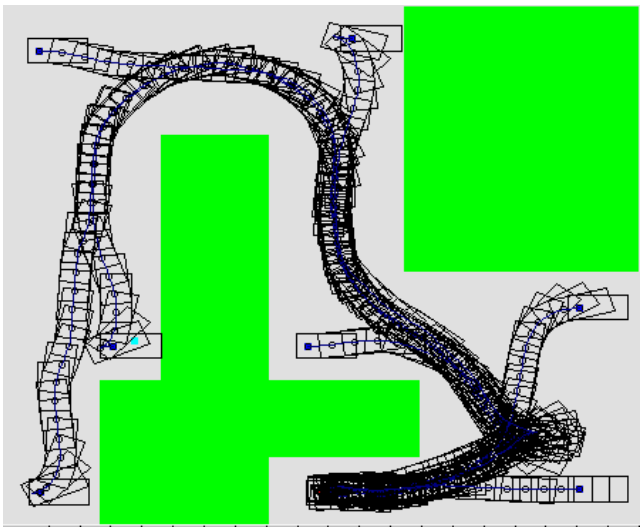

Fig. 3. Optimal motion of the car-like vehicle in the presence of obstacles (RL D-2).

In Fig. 2 are observed the trajectories described by the vehicle during the learning process. In approximately 30 seconds the vehicle is able to reach the goal. The optimal policy is completely updated each 5 seconds.

Fig. 3 shows how the vehicle is able to reach the final position (objective cell) starting from an initial position. Only some trajectories are shown in Fig.3, which correspond to several extreme positions. Note that $\theta = -\pi$ and $\theta = \pi$ are considered two different orientations, e. g., if $\theta = \pi$ the vehicle only can turn clockwise. This restriction can easily be removed to allow the vehicle to select the best path.

Figure 4 shows the distribution of trajectory times according to two adjoining distances and the best result of Q-learning. As we can see, the best result is provided by RL D-2 (99.31%, 20.09 sec). With RL D-k it is not necessary to adjust the transition time such as in Q-learning to get an optimal behaviour of the vehicle. Moreover, the transition time is not known in advance and it should not be fixed in the entire state space, especially in complex dynamic systems.

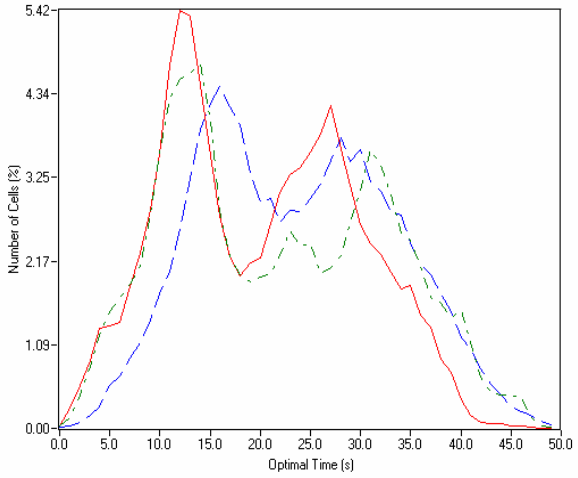

Fig. 4. Distribution of trajectory times with a discretization of 41 cells per axis for RL D-2(red solid line), RL D-3(blue dash line) and Q-learning Ts = 0.9 (green dash dotted line).

VII. Experimental Results

The reinforcement learning controllers have been implemented successfully on the real robot, as shown in Figure 5. Due to the small steering range on the real vehicle ($-21^\circ \leq \zeta \leq 21^\circ$), the best performance in the experiments has been achieved with RL D-3 (41 cells/axis), the controller with the longest adjoining distance. This controller can discriminate between small variations in the steering angle, since longer transitions prevent the robot to fall in cells belonging to the straight direction. In contrast, employing short transitions the trajectory deviation is not enough to reach cells in the transversal direction.

In order to reduce the computation time the steering actions have been limited to three (-21°, 0°, 21°). The region of interest is a small rectangle (2.8×3.6 meters) with obstacles, so the robot needs to go back quite often to access any state in the region of interest. Figure 5 shows a trajectory where the robot has to move between the opposite extremes of the rectangle. The vehicle executes a time-optimal motion going back at the beginning and then going forward avoiding the obstacles until the goal is reached.

VIII. Conclusion

The new RL approach has been successfully employed for optimal motion planning of a real robot. In contrast with conventional RL techniques (Q-learning, Dyna-Q,

Prioritized Sweeping), the algorithm does not need to use function interpolation to find a close to optimal behaviour in continuous state spaces. Furthermore, this approach provides a closed-loop solution in the presence of obstacles including forward and backward motion. The result is a good approximation of the global optimal solution through the adequate selection of the adjoining distance. The technique is robust to noise and changes in the vehicle parameters, since the robot model is estimated on-line updating the optimal motion law in real time.

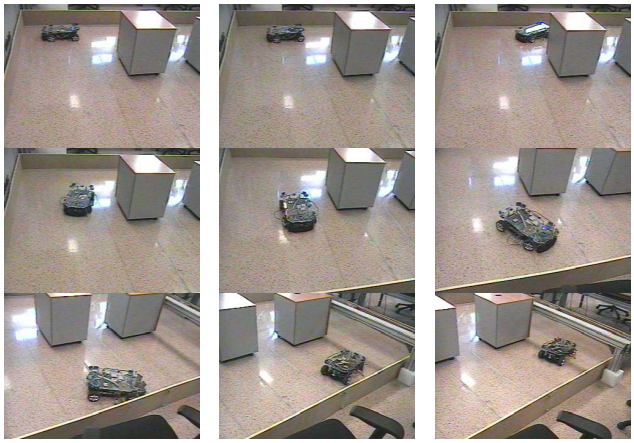

Fig. 5. Sequence of images showing a trajectory learned on the real robot.

REFERENCES

[1] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic, 1991.

[2] J. A. Reeds and R. A. Shepp, "Optimal path for a car that goes both forward and backward", *Pacific J. Math*., vol. 145, no. 2, pp. 367-393, 1990.

[3] D. Gu and H. Hu, "Neural Predictive Control for a Car-like Mobile Robot", *Int. J. of Robot. and Autonomous Systems*, vol. 39, no. 2-3, 2002.

[4] F. Lamiraux and J. P. Laumond, "Smooth Motion Planning for Car-Like Vehicles", *IEEE Trans. Robot. Automat*., vol. 17, no. 4,pp. 498-502, 2001.

[5] T. Fraichard and J. M. Ahuactzin, "Smooth Path Planning for Cars", *IEEE Int. Conf. on Robot. Automat*., pp. 21-26, Seoul, 2001.

[6] J. Barraquand and J. C. Latombe, "On nonholonomic mobile robots and optimal maneuvering", Revue d'Inteligence Artificielle, vol. 3, no. 2, pp. 77-103, 1989.

[7] P. J. Zufiria and T. Martínez-Marín, "Improved Optimal Control Methods based upon the Adjoining Cell Mapping Technique", *J. Optimization Theory and Applications*, *vol. 118(3)*, pp. 657-680, 2003.

[8] T. Martínez-Marín and T. Duckett, "Fast Reinforcement Learning for Vision-Guided Mobile Robots," in *Proc. Int. Conf. Robotics and Automation.,Barcelona, 2005*.

[9] R. Sutton and A. Barto, *Reinforcement Learnin: An Introduction*, MIT Press, 1998.

[10] Bertsekas, *Neurodynamic Programming*, Athena Scientific, 1996.

[11] A. Moore and C. Atkeson, "Prioritized Sweeping: Reinforcement learning with less data and less time", *Machine Learning*, *vol. 13*, pp. 103-130, 1993.

[12] C. S. Hsu and R. S. Guttalu, "An unravelling algorithm for global analysis of dynamical systems: an application of cell-to-cell mapping", *J. Appl. Mech., 47*, 940-948, 1980.

[13] C. S. Hsu, "A discrete method of optimal control based upon the cell state space concept", *J. Optimization Theory and Applications, vol. 46*(4), 1985.

[14] P. J. Zufiria & R. S. Guttalu, "The adjoining cell mapping and its recursive unraveling, Part I: Description of adaptive and recursive algorithms", *Nonlinear Dynamics, 4*, pp. 204-226, 1993.

[15] M. Papa, H. M. Tai, and S. Shenoi, Cell Mapping for Controller Design and Evaluation, *IEEE Control Systems Magazine*, pp. 52-65, April, 1997.