Planning Trajectories in Dynamic Environments Using a Gradient Method

Alessandro Farinelli and Luca Iocchi

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198, Roma, Italy
{farinelli,iocchi}@dis.uniroma1.it

Abstract. In this article we propose an extension for a path planning method based on the LPN-algorithm to have better performance in a very dynamic environment. The path planning method builds a navigation function that drives the robot toward the goal avoiding the obstacles. The basic method is very fast and efficient for a robot with few degrees of freedom. The proposed extension integrates the obstacle dynamics in the planning method to have better performances in very dynamic environments. Experiments have shown the effectiveness of the proposed extension in a very dynamic environment, given by RoboCup soccer matches.

1 Introduction

Path planning is a fundamental task for autonomous mobile robots. Every application that involves the use of an autonomous mobile robot has to deal with path planning and obstacle avoidance. Examples of such applications are: air or navy traffic control, exploration and work in hostile environment (such as sub-sea or space), people rescue during disaster or dangerous situation, office work, robotic soccer, etc. Path planning is a critical task for the success of those applications and many different approaches can be found in literature to tackle the problem [16]. Well known techniques based on road-map construction, cell decomposition, or artificial potential fields, are widely used.

In many applications the robots have to cope with a dynamic environment, in which the problem of path planning and obstacle avoidance becomes much harder, because the robots have to take into account that configuration of the work space changes as time flows. Among other application domains, RoboCup [13] is a very good experimental field for such applications because the RoboCup competitions are characterized by a highly dynamic and hostile environment.

Different solutions for path planning in dynamic environments have been proposed in literature. A first group of methods do not take into account any explicit representation of time and are mostly focussed on extending or adapting a standard path planning method proposed for static environments. The goal is to have a very fast algorithm to plan trajectories in a static obstacle configuration and replan the trajectories at a fixed time interval to take into account

D. Polani et al. (Eds.): RoboCup 2003, LNAI 3020, pp. 320–331, 2004.

environmental changes [14, 18, 3, 15, 10]. Another group of works is based on explicitly considering the time dimension during the path planning process. Some of those works rely on the assumption of knowing in advance the dynamic evolution of the environment [9, 12]. As the complete knowledge of the dynamics of the environment is not available in most cases, other authors have studied the possibility of predicting the future evolution of the environment to plan the trajectories [20, 17, 21, 22].

In this paper an approach to the problem of path planning in dynamic environments is proposed[1]. The general idea is to integrate the obstacle's dynamics characteristic into the planning method, by changing their representation in the robot's configuration space. This is a general approach already used in literature, [9, 20], and can be used with different trajectory planning methods. The effectiveness of the approach depends on the specific planning method, and relies on how the information about the obstacles dynamics are taken into account.

The basic approach, from which the present work has started, is described in [14] and provides for an efficient implementation of a path planning method based on numerical artificial potential field similar to [23, 11]. The method in [14] (referred in this paper as LPN) has very good performance for a robot with few degrees of freedom. In particular this method computes a navigation function that avoids local minima, resulting in very effective trajectories. However by not considering the dynamics of the obstacles often the method results in undesired behavior of the robot when facing moving obstacles. Our extended method (LPN for Dynamic Environment, or LPN-DE) gives us better results in highly dynamic environments as compared with the LPN, but still keeps a low computational time and avoids local minima. The LPN and the LPN-DE algorithms have been implemented and compared with several experiments both in a simulated environment and with real robots in the RoboCup environment.

The LPN method is presented in Section 2 while its extension LPN-DE is described in Section 3. In Section 4 experimental results are given while in Section 5 an analysis of the related works is presented. Finally, conclusions are drawn in the last section.

2 LPN Gradient Method

In this section we describe the method presented in [14] (Linear Programming Navigation gradient method or LPN), while in the next section we describe the extension that we propose to improve its performances when the dynamics of the environment are taken into account.

The LPN gradient method is based on a numerical artificial potential field approach. The method samples the configuration space and assigns a value to every sampling point using a linear programming algorithm. The values for the sampling points are the sampled values of a navigation function; this means that

[1] This research is partially supported by MIUR (Italian Ministry of Education, University and Research) under project RoboCare (A Multi-Agent System with Intelligent Fixed and Mobile Robotic Components).

the robot can find an optimal path, just following the descendent gradient of the navigation function to reach the goal. The method can take as input a set of goal points, the final point of the computed path will be the best one with respect to the cost function discussed below. In order to build the navigation function a path cost must be defined. A path is defined as an ordered set of sampling points

$$P_n = \{p_n, p_{n-1}, ..., p_0\} \tag{1}$$

such that:

- $p_i \in \mathcal{R}^2$
- $\forall\, i = n, ..., 1\, p_i$ must be adjacent to p_{i-1} or along the axis of the work space or in diagonal
- $\forall\, i, j$ if $i \neq j$ then $p_i \neq pj$
- p_0 must be in the set of the final configurations
- $\forall\, i = n, ..., 1\, p_i$ must not be in the set of the final configurations

Given a point p_k, a path that starts from p_k and reaches one of the final configurations p_0 will be represented as $P_k = \{p_k, ..., p_0\}$. A cost function for a path P is an arbitrary function $F : \mathcal{P} \mapsto \mathcal{R}$, where $\mathcal{P}$ is the set of paths. This function can be divided into the sum of an *intrinsic cost* due to the fact that the robot is in a certain configuration, and an *adjacency cost* due to the cost of moving from one point to the next one:

$$F(P_k) = \sum_{i=0}^{i=k} I(p_i) + \sum_{i=0}^{i=k-1} A(p_i, p_{i+1}) \tag{2}$$

where A and I can be arbitrary functions.

Normally, I depends on how close the robot is to obstacles or to "dangerous" regions, while A is proportional to the Euclidean distance between two points.

The value of the navigation function N_k, in a point p_k, is the cost of the minimum cost path that starts from that point:

$$N_k = \min_{j=1,...,m} F(P_k^j) \tag{3}$$

where P_k^j is the j-th path starting from point p_k and reaching one of the final destinations and m is the number of such paths.

Calculating the navigation function N_k for every point in the configuration space directly, would require a very high computational time, even for a small configuration space. The `LPN` algorithm [14] is used to efficiently compute the navigation function. It is a generalization of the `wavefront algorithm` [19, 6] that is based into three main steps:

- assign value 0 to every point in the final configuration and an infinite value to every other point;
- put the goal set points in an *active list*;
- at each iteration of the algorithm, operate on each point of the active list, removing it from the list and updating its 8-neighbors (Expansion phase).

The expansion phase is repeated until the *active list* is not empty. To update a point p we operate as follow:

- for every point q in the 8-neighbors of p compute its value, adding to the value of p the moving cost from p to q and the intrinsic cost of q.
- if the new value for q is less than the previous one, update the value for q and put it into the *active list*.

In Figure 1 an update step is shown.

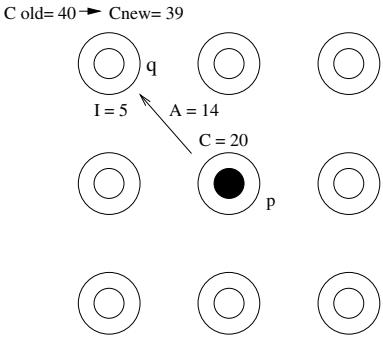

Fig. 1. Update step

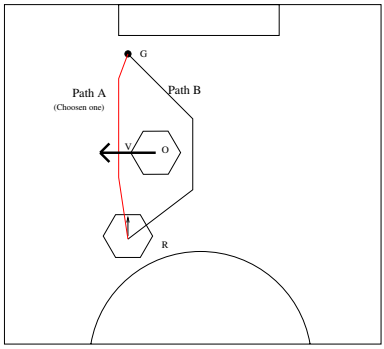

Fig. 2. Problems due to dynamic obstacle

The navigation function is computed according to the intrinsic cost of the sampling points in the configuration space. Suppose the obstacles in the workspace are given by a set of obstacle sampling points. Let $Q(\cdot)$ be a generic function and $d(p)$ the Euclidean distance for the sampling point p from the closest sample point representing an obstacle, then $I(p) = Q(d(p))$. In order to compute $d(p)$ for every sampling point we may apply the `LPN` algorithm giving the obstacle sampling points as the final configuration set, and assigning in the initialization phase a value 0 to the intrinsic cost for all the sampling points. Once $d(p)$ (and then $I(p)$) is computed for every sampling point p, we can execute again the `LPN` algorithm to compute the navigation function.

This method has been chosen for our extension for several reasons. First of all the method is very fast: it is possible to apply the trajectory planning every 100 ms on a grid of 100×100 sampling points on a modest CPU (Pentium 266 MHz)[14]. Moreover, the method finds an optimal path with respect to the cost function used. Those properties are very interesting for the RoboCup environment, which is the environment that we used for our experiments. Thanks to the low computational requirement the sampling of the configuration space can be exploited in order to reach a satisfying precision (10 cm in our case). On the other hand, the known information about the environment (such as field shape and position of fixed obstacles) can be easily and effectively taken into account. Moreover, the intrinsic cost function can be tailored in a very direct

and simple manner, to obtain different trajectories, and so different behaviors of the robot.

However, the LPN method described in this section does not take into account the dynamics of the environment. This is a limitation when coping with very dynamic obstacles and it can result in non-optimal paths in some critical situations. As an example, in Figure 2 we can see that for the robot R to reach the goal G, path B would be better than path A, when the obstacle dynamics is considered, but the LPN method would choose path A, that is the optimal trajectory with respect to the current time frame.

3 LPN-DE Method

The extension to the LPN method for application in Dynamic Environments (LPN-DE) is based on additional information about moving obstacles, which are represented by a velocity vector. Observe that we do not require to have complete knowledge of the dynamics of the environment in advance (that would not be possible in many cases), but only assume to know an estimation of velocity vectors for the obstacles. Notice also that this requirement can be obtained by an analysis of sensor data, and its precision is not critical for the method, that is robust with respect to errors in evaluating such velocity vectors. Moreover, the proposed extension does not depend on the technique used to compute the velocity vectors for the obstacles. How those information are obtained, mainly depends on specific sensors of the robot and our particular implementation will be described later. By suitably taking into consideration the information about the velocity vectors in the planning method we show that our extension gives better result than the basic LPN algorithm.

In order to clarify how the LPN-DE integrate the obstacle dynamics information into the planning process, we introduce the concept of *influence region* of an obstacle. This definition is based on the fact that the intrinsic cost $I(p)$ is generally limited, i.e. it is 0 for points p such that $d(p) > \delta$, for a threshold δ.

Definition 1 *The* influence region *for an obstacle O_i is the set of sample points p whose intrinsic cost $I(p)$ is modified by the obstacle O_i.*

Basically the influence region of an obstacle O_i is the area surrounding the obstacle that we want to avoid, i.e. the set of point around O_i for which $I(p) > 0$. Considering the obstacle pointwise, an intrinsic cost function depending only on the distance $d(p)$, will result in circular *influence regions*, as it can be seen in Figure 3. The LPN-DE extension that we propose here defines a function $I(p)$ resulting in *influence regions* as shown in Figure 4.

As intuitively shown in Figures 3 and 4, the LPN-DE extension is designed in such a way to prefer trajectories for which the probability of collisions (or trajectory intersections) with moving obstacles is minimum. In fact, the function $I(p)$ in LPN-DE modifies the shape of the *influence region* according to the information about the obstacle dynamics, as shown in Figure 4. Consequently

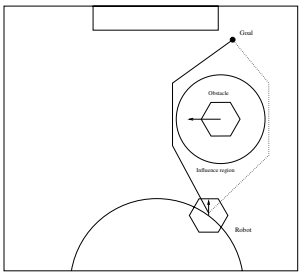

Fig. 3. Simple influence region (LPN)

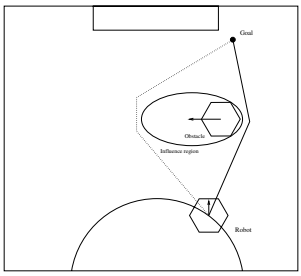

Fig. 4. Extended influence region (LPN-DE)

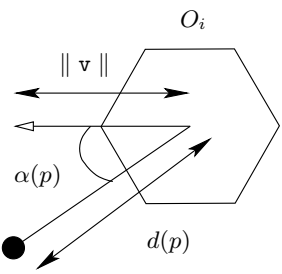

Fig. 5. Values for computing $I(p)$

the behavior of the robot is more adequate to the situation, since the chosen path will be the one passing behind the other robot, which is shorter an safer.

If a given point p belongs to the influence region of an obstacle O_i the intrinsic cost function for p will depend not only on the distance from O_i, but also on the position of p with respect to the velocity vector of O_i.

Therefore the function $I(p)$ will depend on (see Figure 5):

- the distance $d(p)$ of the point p with respect to O_i
- the angle $\alpha(p)$ between the velocity vector and the line reaching the center of O_i,
- the velocity's module $\| \mathtt{v} \|$ of O_i

So $I(p)$ in LPN-DE is given by a function $Q(d(p), \alpha(p), \| \mathtt{v} \|)$.

To compute the intrinsic cost function $I(p)$, depending on the values described above, the LPN algorithm has been slightly changed. We represent each moving obstacle as a circle and we use the robot sensors for obtaining, at every new planning cycle, (an estimation of) the center, the radius and the velocity vector of each obstacle. The basic structure of the algorithm is the same as described in Section 2, but the intrinsic cost function, the initialization and update steps for computing the intrinsic cost change. Specifically:

1. $I(p) = Q(d(p), \alpha(p), \| v \|)$.
2. In the initialization step we have to represent the velocity vector (module and heading) of each obstacle in each of the sampling points representing the center of an obstacle; we give to each of those point a high value and put to zero the values of all the other sampling points.
3. In the update step, if the new computed cost for a point q (I(q)) adjacent to a point p is greater then the actual cost:
   - update the cost of the point q,
   - propagate the velocity vector and position of the obstacle center,
   - put q in the *active list*.

In particular at each step we need to propagate the velocity vector and position information in all the points that are part of the influence region of the obstacle because those information are needed in the computation of the intrinsic cost function I(p).

3.1 Implementation

The described LPN-DE method has been implemented on different kinds of mobile robots, and several experiments in the RoboCup Middle-Size environment [13] has been performed.

In our implementation the function $Q(d(p), \alpha(p), \| v \|)$ is:

$$\begin{cases} C & if\ d(p) \leq D \\ C - F(\alpha(p), \| v \|)d(p) & if\ d(p) > D \text{ and } F(\alpha(p), \| v \|)d(p) < C \\ 0 & if\ d(p) > D \text{ and } F(\alpha(p), \| v \|)d(p) \geq C \end{cases}$$

where

- C is the intrinsic cost of a sample point inside an obstacle.
- D is the radius of the augmented obstacle.
- $F(\alpha(p), \| v \|) = \alpha(p) * \beta * \left(\frac{1}{\|v\|} + \gamma\right)$

For higher value of $\alpha(p)$ the value of $F(\alpha(p), \| v \|)$ becomes higher, so that the influence region results stretched along the axes of the obstacles velocity vector. For lower value of $\| v \|$ the value of the function $F(\alpha(p), \| v \|)$ becomes higher, so that the influence region is greater for greater values of the obstacles velocity module. The function has been chosen because it is a very simple function that gives us a reasonable shape for the influence region (similar to the one represented in figure 4). The value of the parameters have been tuned for the specific application.

4 Experiments and Results

In this section we present some experiments made in order to show the improvement in the performance of the LPN-DE method. The experimental environment has been the RoboCup Middle-Size field installed in our laboratory, that is slightly reduced with respect to current rules. For the implementation of the method we have used sampling rate of 100 mm. The main robot sensor is a color camera and the experiments have been performed on different kinds of robots, with different kinematics (unicycle-like and holonomous ones). The method has also been used by the SPQR middle-size team during the German Open 2002.

We have performed two kinds of experiments: the first one is based on a simulation environment, in which we have collected a set of quantitative data; the second kind of experiments has been done on real robots, substantially confirming the behaviors obtained in simulation.

In the simulation environment we made the assumption that the robots know in every instant the current positions and the velocity vector of the obstacles, in the experiments with real robots those information have been extracted by software vision modules from images acquired by the color camera.

4.1 Experimental Setting

Figures 6 and 7 present a typical situation occurred in all the experiments we have performed. The robot should reach a goal point, while a moving obstacle crosses the planned trajectory. In Figure 6 the robot uses the LPN method to plan the trajectories while in Figure 7 the LPN-DE one. The black dots in the figures represent sample points influenced by the moving obstacle, the lines starting from the robot represent the planned trajectory at that instant. The trajectory planned in Figure 7 using the LPN-DE method is better than that planned in Figure 6 as the following experiments show.

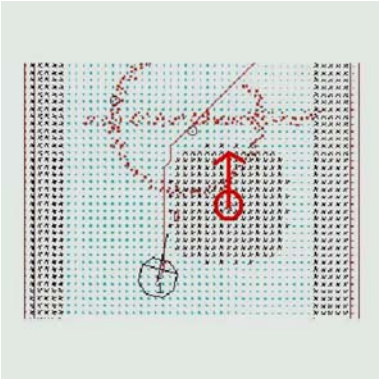

Fig. 6. LPN method

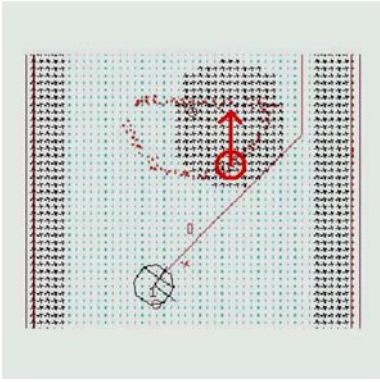

Fig. 7. LPN-DE method

4.2 Experiments with a Simulator

In this set of experiments the robot has to reach a predefined set of check points in sequence, while a simulated obstacle follows trajectories that intersect many times the standard path of the robot. An example of such experiments is given in Figure 8, in which the robot has to reach iteratively the check points highlighted with circles, while a moving obstacle moves along the bold path.

This kind of experiments have been performed by measuring the average time needed for the robot to reach the next point in the sequence, considering different velocities for the robot and the obstacles and comparing the LPN method and its extension LPN-DE. The table in Figure 9 shows the average time (from a 15 minutes execution of the experiment) needed to reach the next check point, for different velocities of the robot and of the obstacle and comparing the values of LPN (upper cells) with LPN-DE (lower cells). The values in the table shows that LPN-DE generally performs better than LPN, since it is generally able to avoid critical situations in which the robot trajectory intersects the obstacle path.

4.3 Experiments with Real Robots

The second set of experiments has been done on real robots by considering two robots that, in order to reach two different target positions, must plan trajec-

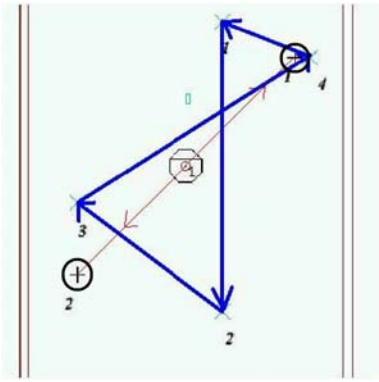

Fig. 8. Simulation experimental setting

	Vel Max Robot 800 mm/sec	Vel Max Robot 1600 mm/sec
Vel Max Obst. 1000 mm/sec	6.3 sec	5.9 sec
	5.8 sec	5.6 sec
Vel Max Obst. 2000 mm/sec	6.5 sec	6.2 sec
	6.3 sec	5.9 sec

Fig. 9. Simulation experimental results

tories that intersect each other. Also in this case we have evaluated the average time for reaching the two target points and we have obtained results similar to the ones presented in the previous section, in which the LPN-DE algorithm generally performs better.

In fact a typical situation arising in this experimental setting is reported in Figures 10 and 11, that represent the actual trajectories followed by the two robots, whose initial positions S_1 and S_2 are shown in the left side of the figures and the target points G_1 and G_2 are in the right side.

In the first case both the robots use the LPN method, and their behavior is not optimal since one robot (robot 1) always tries to pass in front of the other one. The second case, in which both the robots use the LPN-DE method, shows instead that the additional information about the obstacle dynamics has been properly exploited in the LPN-DE method for planning a better trajectory in presence of moving obstacles. Notice also that in any case the trajectories in Figure 11 are not optimal, as they would be if any robot would know in advance the trajectory planned by the other. Thus the robots execute a small portion of their path in parallel as in the previous case. However, in this case, at a certain point one of the robots (specifically robot 1) is able to detect such situation and decides to pass behind the other robot.

5 Related Work

Among the several techniques proposed in the literature for path planning in dynamic environments, we have mainly compared our approach to approaches that have been experimented in the RoboCup domain, both because we choose it as our testbed, and because in this domain a solution to the problem of obstacle avoidance and path planning among moving obstacle is particularly needed.

The work described in [8] presents an approach based on modified potential field. The method focuses on the generation of low level commands that enable the robot to dribble the ball amongst moving obstacle, but do not address the

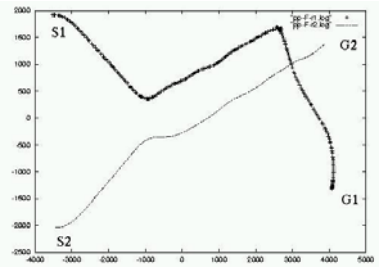

Fig. 10. LPN method

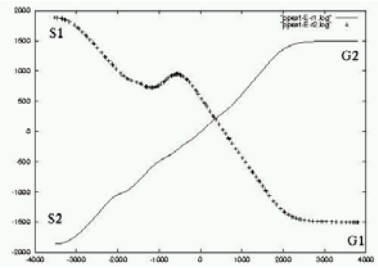

Fig. 11. LPN-DE method

problem of local minima, that is very important in a crowded dynamic environment, like RoboCup soccer; where often several robots are close to the ball. Our approach on the other hand, as already pointed out, generates navigation functions that does not contains local minima.

The problem of local minima is instead addressed in [4]. where the authors propose a method based on a combination of random exploration and biased motion towards the goal configuration. This method do not solve completely the problem of local minima, but reduces their occurrence.

A different approach is presented in [5], this work is focused on an hybrid path planning method that choose the best path planning algorithm depending on the specific situation. Simulation experiments show that the hybrid method outperform all the simple methods, but no guarantee is given on the optimality of the chosen trajectory.

Finally, [1] presents an adaptive planner, based on a static planner (developed in [2]), that tries to slightly change the plan while the environment configuration evolves. The experiments reported show good results for this approach; however, the time required for the trajectory generation is not constant with respect to obstacle configuration, and in particular becomes quite high in particular situations (e.g. when the adaptation of the planned trajectory fails).

With respect to all the discussed approaches the present work is a novel attempt to exploit the obstacles dynamic inferred from the environment during the task execution. In fact, the LPN method is guaranteed to plan optimal trajectories with low computational time requirements in real-time and by avoiding local minima. Our LPN-DE extension maintains all the above advantages of the LPN method (including low computational time), but performs much better in highly dynamic situations.

The general approach of integrating the obstacles dynamic information into the path planning procedure, changing the obstacle configurations accordingly, could be also extended to other path planning methods; as an example a suitable method for an effective extension based on this approach is the one described in [7]. In this work harmonic functions are used in order to compute a free path for a robot, using a potential field approach. Those functions contain very interesting properties, such as completeness in the path calculation, robustness to unanticipated obstacles and rapid computation.

6 Conclusions

In this article we have presented an approach to cope with dynamic environments extending a very efficient method for path planning presented in [14]. This method has been chosen because it is appropriated for our testing environment (RoboCup). The basic idea of the LPN-DE relies on the integration of the information about the velocity of moving obstacles into the path planning algorithm. In particular, in the present implementation, the dynamics of the obstacles have been integrated in the LPN method, modifying the cost function used for computing the global navigation function. Although this extension does not guarantee optimal trajectories along the time dimension, the performed experiments show that the LPN-DE method performed better than LPN, when facing fast moving obstacles, keeping the same computational requirement.

The LPN method and the extension have been implemented on robotic platforms with different kinematic models (unicycle-like and holonomous). As the present work is focused on the generation of effective trajectories for dynamic environments, we did not investigate the issues related to the low level control of the robotic platform. We rather provided a simple implementation of a low level robot controller that given the desired trajectory and the robot kinematic model, is able to drive the robot along the trajectory fulfilling the application constraints, that in the RoboCup case are high speed and reasonable precision.

The reported experiments show that better results can be obtained by integrating into the path planning method the information on the obstacles dynamics, and that the proposed extension in highly dynamic environments turned out to be effective and it has actually improved the performance of the robotic platform. As future work, we are currently investigating the possibility of applying a similar extension for dealing with moving obstacles to other path planning methods.

References

1. J. Baltes and N. Hildreth. Adaptive path planner for high dynamic environment. *RoboCup 2000: Robot Soccer World Cup IV*, 2000.
2. A. Bicchi, G. C. Casalino, and Santilli. Planning shortest bounded-curvature paths for a class of nonholonomic vehicles among obstacles. *Proocedings of the IEEE Int. Conference on Robotics and Automation*, 1995.
3. J. Borenstein. The vector field histogram-fast obstacle avoidance for mobile robots. *Robotics and Automation, IEEE Transactions on*, Volume: 7 Issue: 3:278 –288, 1991.
4. J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *In Proceedings of IROS-2002*, Switzerland, October 2002.
5. S. Buck, T. Schmitt, and M Beetz. Planning and executing joint navigation tasks in autonomous robot soccer. In *In RoboCup Int. Symposium*, Seattle, 2001.
6. Arkin R. C. Integrating behavioral,perceptual and world knowlodge in reactive navigation. In *Robotics and Autonomus Systems 6*, pages 105–122, 1990.
7. C. I. Connolly and R. A. Grupen. On the application of harmonic functions to robotics. *Journal of Robotic Systems*, 10(7):931–946, 1993.

8. B. Damas, L. Custodio, and P. Lima. A modified potential fields method for robot navigation applied to dribbling in robotic soccer. In *Proc. of RoboCup 2002 Symposium*, Fukuoka, Japan, 2002.
9. P. Fiorini and Z. Shiller. Motion planning in dynamic environments using the relative velocity paradigm. *IEEE Int. Conference on Robotics and Automation*, vol.1:560 –565, 1993.
10. A. Fujimori, P. N. Nikiforuk, and M. M. Gupta. Adaptive navigation of mobile robots with obstacle avoidance. *Robotics and Automation, IEEE Transactions on*, Volume: 13 Issue: 4:596 –601, 1997.
11. R. A. Jarvis. Collision-free trajectory planning using the distance transforms. *Mechanical Engineering Transaction of the Institution of Engineers*, ME10(number 3):187–191, 1985.
12. R. Kindel, D. Hsu, J.C. Latombe, and S. Rock. Kinodynamic motion planning admist moving obstacle. *Proocedings of the 2000 IEEE Int. Conference on Robotics and Automation*, 2000.
13. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for AI and robotics. In *Lecture Note in Artificial Intelligence*, volume 1395, pages 1–19, 1998.
14. Kurt Konolige. A gradient method for realtime robot control. *AIROS*, 2000.
15. B. Kreczmer. Application of parameter space discretization for local navigation among moving obstacles. *Robot Motion and Control, 1999. RoMoCo '99. Proceedings of the First Workshop on*, pages 193 –198, 1999.
16. J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publisher, 1991.
17. J. Miura and Y. Shirai. Modelling motion uncertainty of moving obstacles for robot motion planning. In *IEEE Int. Conference on Robotics and Automation, Proceedings. ICRA '00*, volume Volume: 3, pages 2258 –2263, 2000.
18. G. Oriolo, G. Ulivi, and M. Vendittelli. Path planning for mobile robots via skeleton on fuzzy maps. *Intelligent Automation and Soft Computing*, 1996.
19. S. Thrun, A. Buken, W. Burgarg, D. Fox, T. Frohlinghaus D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in RHINO. In *AI-based Mobile Robots : Case studies of successful robot systems*. MIT Press, Cambridge,MA, D. Kortenkamp, R.P. Bonasso, and R. R. Murphy editors, 1998.
20. M. Yamamoto, M. Shimada, and A. Mohri. On-line navigation of mobile robot under the existence of dynamically moving multiple obstacle. In *Assembly and Task Planning, 2001, Proceedings of the IEEE Int. Symposium on*, 2001.
21. Huiming Yu and Tong su. A destination driven navigator with dynamic obstacle motion prediction. *Proceedings of the IEEE Int. Conference on Robotics and Automation*, 2001.
22. N. H. C. Yung and C. Ye. Avoidance of moving obstacles through behaviour fusion and motion prediction. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE Int. Conference on*, volume Volume: 4, pages 3424 –3429, Oct 1998.
23. A. Zelinsky. Using path transforms to guide the search for findpath in 2D. *IJRR*, 13:315–325, 1994.