XI Reunión de Trabajo en Procesamiento de la Información y Control, 21 al 23 de septiembre de 2005

Route planning for vehicle autonomous navigation, based on geometrical regions. Part I: Single approach point

L. M. Di Matteo[1], A. C. Mangone[2], M. L. Muzzio[3], C. Verrastro[4]

[1] *Artificial Intelligence Group, Univ. Tecnológica Nacional Fac. Regional Bs. As., Buenos Aires, Argentina.*
Leandro.DiMatteo@ieee.org
[2] *Artificial Intelligence Group, Univ. Tecnológica Nacional Fac. Regional Bs. As., Buenos Aires, Argentina.*
sigma1@ciudad.com.ar
[3] *Universidad Argentina de la Empresa, Facultad de Ingeniería, Buenos Aires, Argentina.*
maximiliano@termoacustica.com.ar
[4] *Artificial Intelligence Group, Univ. Tecnológica Nacional Fac. Regional Bs. As., Buenos Aires, Argentina.*
Centro Atómico Ezeiza, Comisión Nacional de Energía Atómica, Buenos Aires, Argentina.
cverra@cae.cnea.gov.ar

Abstract-- **This paper presents an algorithm for trajectory generation for vehicle autonomous navigation.**

This algorithm generates a trajectory from an initial to a goal position, avoiding obstacles, where the goal point moves all the time and also the obstacles, and where the processing time must be as short as possible.

This method gives the possibility to find the optimum path, selecting a determined trajectory from many possible ones.

We model the environment using geometrical regions and it is not grid based, that is why we use algebraic expressions to calculate obstacle free paths. It is assumed that obstacle positions are known all the time.

Keywords-- **Robotics, autonomous navigation, path planning, optimum routing, mobile vehicle navigation.**

I. INTRODUCTION

Control problems involving the autonomous navigation of mobile vehicles have attracted much attention in the last decade.

Algorithmic methods based on potential fields have demonstrated to be the most effective ones (Kathib at al., 1998; Borenstein and Koren, 1991b). These techniques are very stable against sensing errors and easy to implement, but these have some difficulties when obstacles are very near and the obstacle free paths are very thin.

There are several control techniques based on artificial intelligence (Patino and Carelli, 2003) that have to lead with structured or dynamic environments. In structured environments the motion can be planned in advance but in dynamical environments actions must be recalculated in real time.

Others works model the environment on a grid based map (Pereiro and Verrastro, 2003; Weigl at al, 1993) where each cell is occupied by an obstacle or not. This model is very useful to process search algorithms in optimum path searching.

In recent application jobs, we had to lead with the trajectory calculation problem, in a high speed dynamical environment where the goal position moves all the time and also the obstacles.

In this work we propose a simple algorithm that generates a trajectory from an initial to a goal position, avoiding obstacles, where the goal point moves all the time and also the obstacles, and where the processing time must be as short as possible.

We model the environment using geometrical regions and it isn't grid based, that is why we use algebraic expressions to calculate obstacle free paths. It is assumed that obstacle positions are known all the time.

II. OBJECTIVES

The main task of this algorithm is to determine a trajectory to achieve a defined point, avoiding obstacles and arriving to that point with a specified entry angle.

Current and target position must be given, as well as the environment knowledge.

In order to arrive to the target point with the specified entry angle, we use an approximation radius that establish an approaching point.

In conclusion, we need the following input parameters for the system: initial position, final position, xy obstacle points, entry angle and approximation radius.

III. ENVIRONMENT MODELLING

This work propose to work with an environment without a cell oriented structure. So, we use algebraic expressions to model objects and trajectories.

We assume a two dimensional world, and where each obstacle is a square with a known side length.

In case those obstacles are irregular, not squares, the environment must be preprocessed to put it in the squared form.

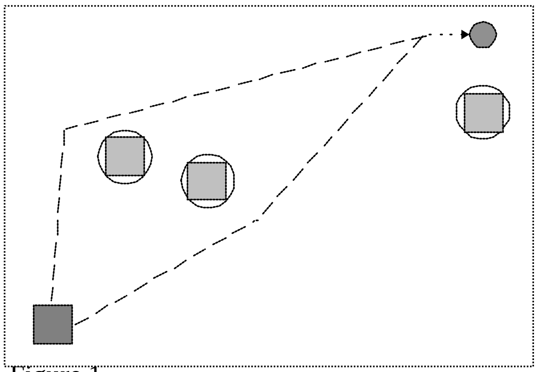
Figure 1.

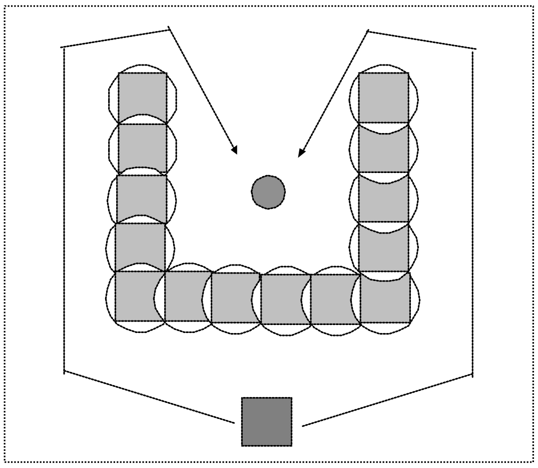
Figure 2.

Once we have a modularized workplace, for each obstacle block the mass center must be calculated, that is to say, the center of the square in a squared obstacle.

Each obstacle block is represented with a circumference, Eq. 1, where the center is the calculated mass center of the object, and its radius is predefined by the user, Fig. 1 and Fig. 2.

$$(X - Xc)^2 + (Y - Yc)^2 = Rrobot^2 \qquad (1)$$

It is intended to configure the obstacles radius a little greater than obstacle diagonal, with the aim of taking into account the mobile robot dimensions, in order to avoid lateral collision between the robot and obstacles.

IV. GENERAL DESCRIPTION

The best way to understand this algorithm, before going to a deep explanation, is using an example, describing a simple and typical situation.

The environment to test is shown in Fig. 3, in which it is possible to see the initial robot position, on the bottom left corner, the target position on the upper right corner, and two squared obstacles in the middle.

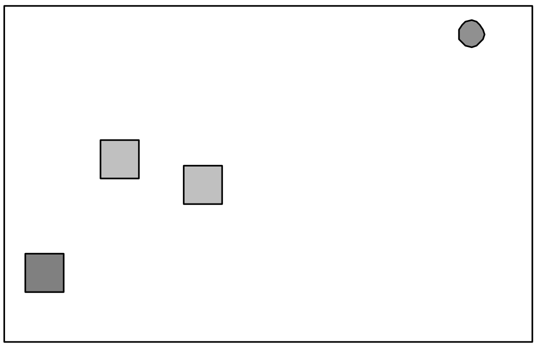
Figure 3.

First at all, approximation point is calculated, Fig. 4.

We use that point as a way point previous to reach the final point with the aim of helping the robot to arrive to the target point with the specified angle.

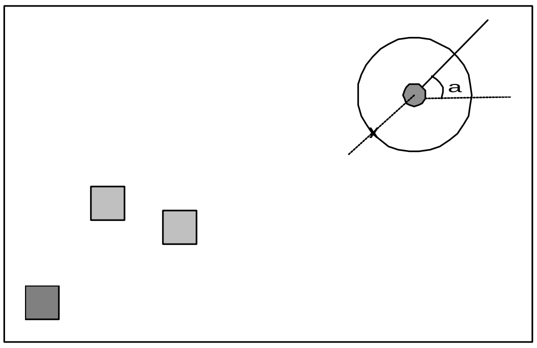

Figure 4.

After that, the route planning begins. The first step is to try going to the approach point directly, Fig. 5, if it is possible it is done, otherwise, like in the figure, two alternatives way points are calculated, one on the robot right hand and the other one on the left hand.

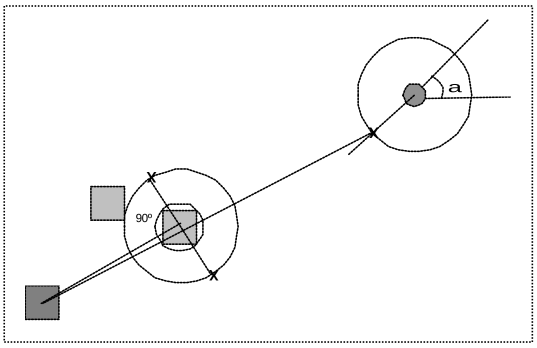

Figure 5.

If the route planner decides to take the point on the robot right hand, Fig. 6, the obstacle free path is checked again, but in this case between the robot and this alternative way point. Like this path is obstacle free, this alternative way point is taken as initial point and the process starts again until the robot reaches its objective point.

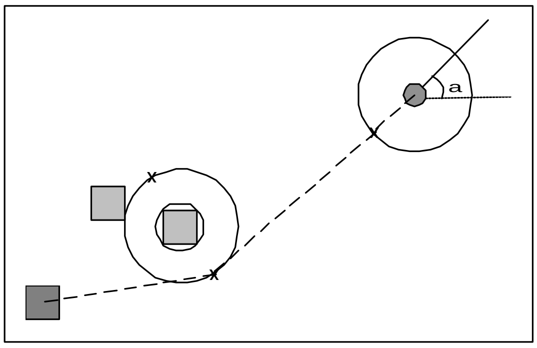

Figure 6.

Lets analyze what was happened if the route planner had decided to take the robot left hand point, see Fig. 4 and then Fig. 7.

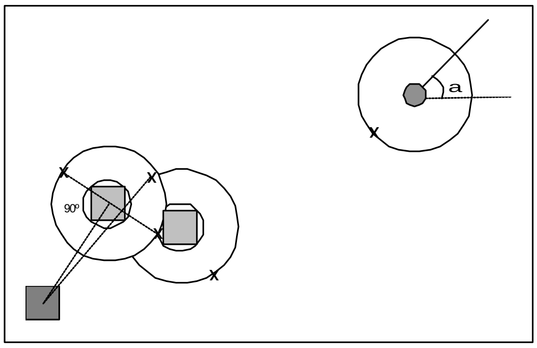

Figure 7.

In that case, when trying to check obstacle free path, a new obstacle is found. Here this waypoint is not taking into account, and two new alternative way points are calculated, in this case these are calculated against the new obstacle. Now suppose route planner selects the way point on the robot left hand, later the obstacle free path is checked. As shown, this path is free, so that way point is taken into account, and is set like initial point for starting the search process again. The process continues to reach the final point, Fig. 8.

Before to tell robot to do a movement, the route planner generates two trajectories, one using valid way points taken from the robot left hand and other one using valid way points taken from the robot right hand. After that, trajectory length are calculated, and it is selected the shorter one.

Note that it is possible to generate a binary tree to select the shortest trajectory, but it is not done in this work.

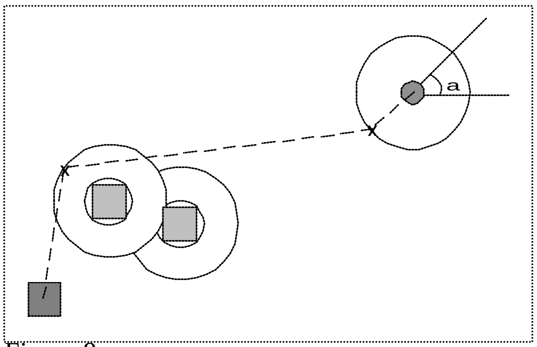

Figure 8.

After route planner decides what trajectory is the best the algorithm returns the first way point of the selected trajectory. Then the robot moves to that xy point.

It is very important to note that the algorithm always return the first way point of the selected path and in each differential time the trajectories are recalculated. It permits to work in a dynamical environment where obstacles are in movement. The current and target point can be in movement, too.

Therefore, this route planner algorithm can be used in static environments as well as dynamical ones.

V. APPROACH POINT CALCULATION

We use the approaching point as a waypoint previous to reach the final point with the aim of helping the robot to arrive to the target point with the specified angle, Fig. 4.

The entry angle is given as a parameter, also an approximation radius, it is used to calculate the approaching point. This radius indicates the distance between the goal point and the approach point.

Being (Xb, Yb) the target point, Rapp the approaching radius, alfa the entry angle. In order to find this point, it is necessary to solve the following equation system:

$$\begin{cases} (X - Xb)^2 + (Y - Yb)^2 = Rapp^2 & (2) \\ Y = m * X + b & (3) \end{cases}$$

where:

$$m = arctg(alfa) \qquad (4)$$

$$b = -m * Xb + Yb \qquad (5)$$

That is, the intersection between the approximation circumference and the straight line equation which intersects the target point and the approaching point.

Now the equation system returns a pair of points, (Xapp1, Yapp1) and (Xapp2, Yapp2), and it is necessary to discriminate which of them is the valid one. In order to determinate the point of interest (Xappr, Yappr) we use this code:

```
////////////////////////////////////////////////////////////////////
Angle = GetAngle(Xapp1,Yapp1, Xb, Yb);
if( alfa == Angle )
    {
    Xappr = Xapp1; Yappr = Yapp1;
    }
else
    {
    Xappr = Xapp2; Yappr = Yapp2;
    }
////////////////////////////////////////////////////////////////////
```

VI. WAYPOINTS SEARCH

First of all, it is necessary to determine if an obstacle is present between the robot and the target point, Fig. 5. A circumference models the obstacle, and the desired trajectory is denoted by a line equation that intersects the robot position and the target point. If the obstacle is located in the middle of the trajectory, the following equation system must have solution in the real domain:

$$\begin{cases} (X - Xb)^2 + (Y - Yb)^2 = Rrobot^2 & (6) \\ Y = m * X + b & (7) \end{cases}$$

Where,

$$m = \frac{Yb - Yr}{Xb - Xr} \qquad (8)$$

$$b = -m * Xr + Yr \qquad (9)$$

Being Xr,Yr the robot position and Rrobot the modeled robot radius. .

If no obstacle is found, the waypoint is the target point.

If the path isn't free of obstacle, two alternative way points are calculated.

The process to calculate these points is as follows. By one hand, we have the straight line that intersects the robot and obstacle positions, Eq. 10, and therefore we can obtain its normal line, Eq. 11.

$$Y = \frac{Yo - Yr}{Xo - Xr} * X + \left[\left(\frac{Yr - Yo}{Xr - Xo} \right) * Xr - Yr \right]$$

(10)

$$Y_\perp = \frac{Xr - Xo}{Yr - Yo} * X_\perp + \left[\left(\frac{Xo - Xr}{Yo - Yr} \right) * Xo - Yo \right]$$

(11)

By the other hand, we have a circumference centered in the obstacle and with a pre assigned radius Rway. Note that Rway must be greater than robot radius Rrobot.

The two way points are obtained from the intersection between the way point circumference and the normal line mentioned above, Eq. 12 and Eq.13.

$$\begin{cases} (X - Xo)^2 + (Y - Yo)^2 = Rway^2 \\ (12) \\ Y = \frac{Yo - Yr}{Xo - Xr} * X + \left[\left(\frac{Yr - Yo}{Xr - Xo} \right) * Xr - Yr \right] \\ (13) \end{cases}$$

Until here, we have the two way points (Xw1,Yw1) and (Xw2, Yw2).

Now is interesting to know which one is located on the robot right hand and which one is located on the robot left hand. This is important for the route planner, it gives the possibility to create a binary tree to make a list of all possible obstacle-free trajectories.

To determine that, case presented in Fig. 8 is analyzed. Then the following programming code must be applied:

```
////////////////////////////////////////////////////////////////////
if((GetAngle(Xr, Yr, Xw1, Yw1)
            – GetAngle(Xr, Yr, Xo, Yo)) > 0)
{
Xder = Xw2; Yder = Yw2; Xizq = Xw1 ; Yizq = Yw1;
}
else
{
Xder = Xw1; Yder = Yw1; Xizq = Xw2 ; Yizq = Yw2;
}
////////////////////////////////////////////////////////////////////
```

Being (Xder, Yder) the point located on the robot right hand and (Xizq, Yizq) the point located on the robot left hand.

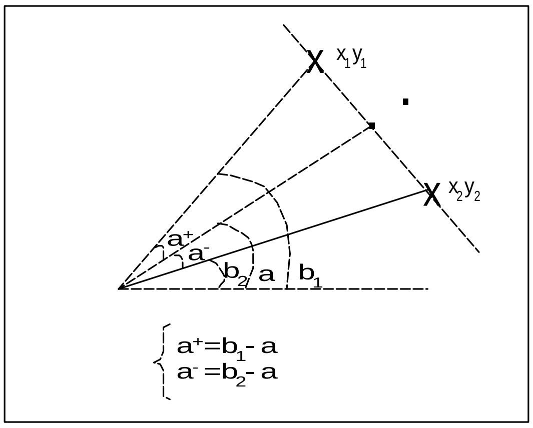

Figure 8.

VII. ROUTE PLANNER

This is the main algorithm. It gives the shortest trajectory without obstacles, searching waypoint by waypoint.

The Fig. 9 illustrate the main body of the algorithm. It is made a list for all way points that conforms a trajectory free of obstacles from the current robot position to the target point.

In our case that sequence is executed twice, the first pass uses all way points located on the robot right hand, and in the second pass the way points located on the robot left hand are taken.

This routine returns two trajectories, that is, two way point lists. The first trajectory denotes the obstacle free path taking all way points placed on the robot right hand and the second one takes all way points placed on the robot left hand.

Here it is possible to construct a binary tree and find the optimum trajectory using searching algorithms. But in our work, we look for the shortest trajectory between the two lists mentioned above, it gives a better processing speed, an important key in real time systems, although it is possible that the selected path not be the shortest one. Only are considered 2 from 2^T possible trajectories, where T is the number of found obstacles.

VIII. EXPERIMENTAL RESULTS

To test our work we use a robot soccer simulation platform, more precisely, the simulator v1.4 used by FIRA organization in its tournaments.

A short code in C++ language was written down to test the algorithm, all data was logged to a text file. Some results are shown in Fig. 10, 11, 12 and 13.

The dark gray box represents the mobile robot, the light gray boxes are the obstacles, in this case these were other players, and the goal point is the ball. In all cases the desired entry angle is 45 degrees.

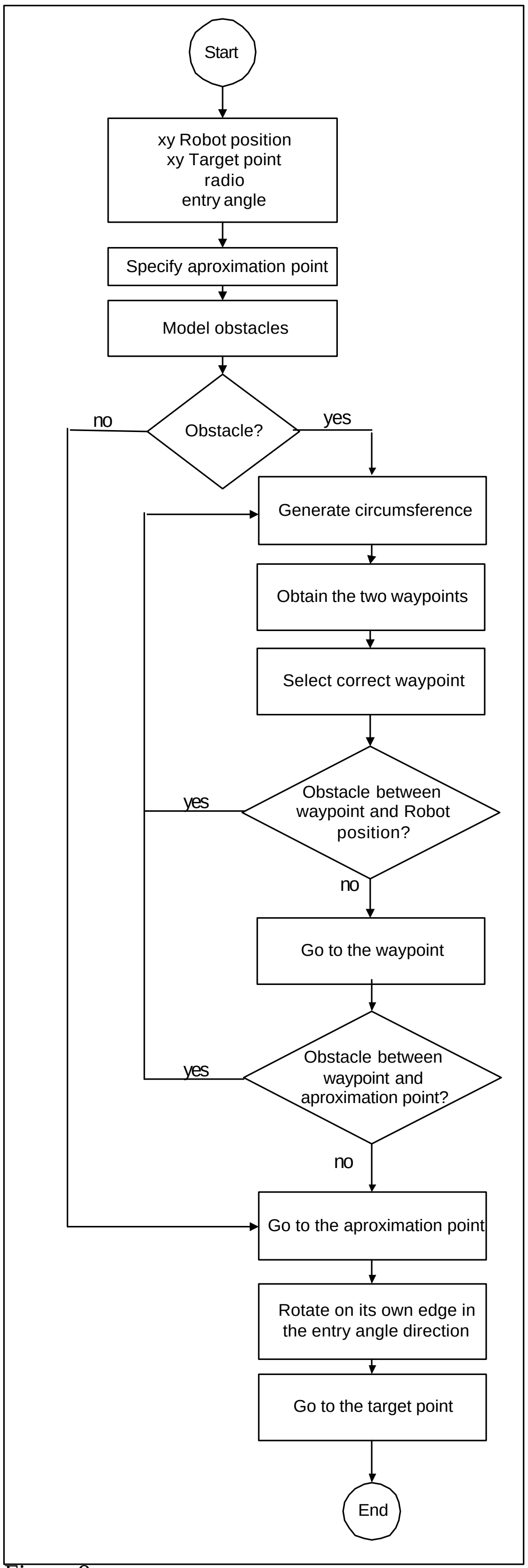

Figure 9.

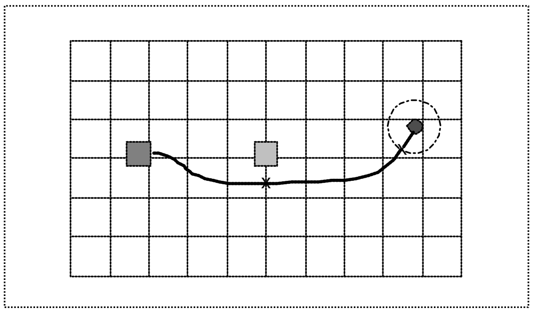
Figure 10.

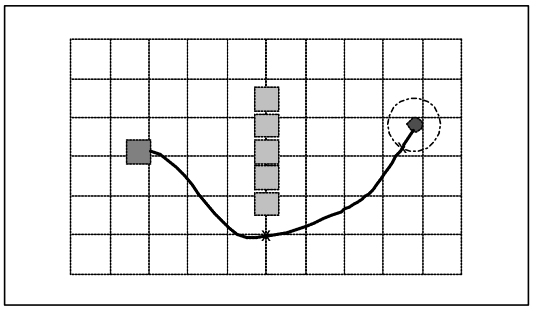
Figure 11.

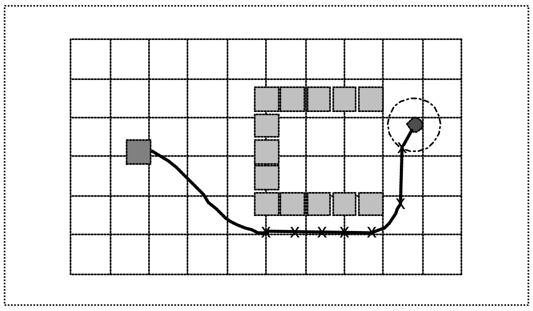
Figure 12.

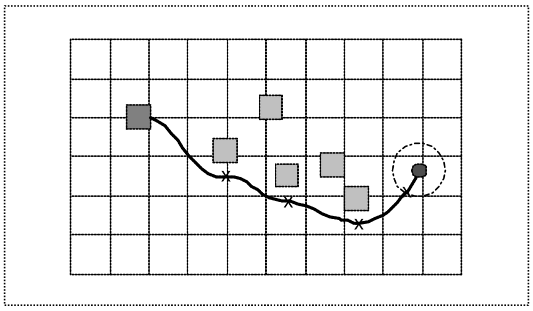
Figure 13.

IX. CONCLUSIONS

As seen in the experimental results, the algorithm proposed achieves its objective successfully, the simulation demonstrated that it fits to static and dynamic environments.

The selection of a "good" path from only two trajectories seems to be very plausible for systems that required a high speed response.

Spite of using the approach point, that helps the mobile vehicle to arrive to the target point with the specified entry angle, some deviations were observed, due to the inertial factor of the mobile vehicles.

Therefore, this paper addresses the problem of trajectories planning in static or dynamic environments when the mobile vehicle has a low inertial behavior, where the inertial factor can be ignored.

That is a reason to continue working in a more advanced algorithm, also efficient for vehicles with a high inertial factor, as ships, airplanes or terrestrial heavy mobile vehicles with high speed movements.

REFERENCES

Alberino S., Folino P. and Verrastro C. "Variante en el algoritmo PID para evitar el uso de un generador de trayectoria trapezoidal " X RPIC Proceedings, San Nicolás, Bs. As., 659-663 (2003).

Borenstein J. and Y. Koren, "The Vector Field Histogram Fast Obstacle Avoidance For Mobile Robots," IEEE Journal of Robotics and Automation, 7, No 3, 278-288 (1991a).

Borenstein J. and Y. Koren, "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation," Proceedings of the IEEE-ICRA, Sacramento, California, 1398-1404 (1991b).

Hwang, Y.K. and N. Ahuja. "Gross Motion Planning – A Survey", ACM Computing Surveys, 24(3), 219-291, (1992).

Khatib, O. "Real-time obstacle avoidance for manipulators and mobile robots" Proceedings IEEE-ICRA, St. Louis MO, 500-505 (1985).

Latombe, J. C. "Robot Motion Planning", Kluwer Academic Pub. Boston (1991).

Orqueda O. and Agamennoni O. "Motion Planning and control of Autonomous robots – I: Generalized Potential Field Functions" X RPIC Proceedings, San Nicolás, Bs. As., 541-546 (2003).

Patiño D. and Carelli R. "Adaptive Critic Design-Based Optimal Control For Mobile Robots Navigation", X RPIC Proceedings, San Nicolás, Bs. As., 503-507 (2003).

Pereiro F. and Verrastro C. "Sistema de Comando y Navegación para Robot Móvil con Arquitectura Distriuida" X RPIC Proceedings, San Nicolás, Bs. As., 565-569 (2003).

Weigl M., Siemiatkowska B., Siroski K., Borowski A.: "Grid-Based mapping for autonomous mobile robot", Robotic and Autonomous Systems, Amsterdam, Holland, (1993).