The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

A Lyapunov-stable, sensor-based model for real-time path-tracking among unknown obstacles

Antonio Sgorbissa, Alessandro Villa, Andrea Vargiu, Renato Zaccaria

Abstract— **The article proposes a feedback control system for real-time navigation and obstacle avoidance that is made of two components: (i) a sensor-based, real-time model that generates and periodically updates the path on–line in order to avoid both known and unforeseen obstacles, and (ii) a feedback-control model that is capable of driving a unicycle vehicle along the collision free path. The system has some unique characteristics, among which it requires very few computational resources as a consequence of its extreme simplicity. In spite of this, it is formally demonstrated to be asymptotically stable, as well as computationally efficient to be implemented in real-world scenarios where obstacles are not known, and possibly move in the environment.**

I. INTRODUCTION

The article proposes a feedback control system for real-time navigation and obstacle avoidance (see other approaches to the problem in [4][5][6]) that is made of two components:

- *Motion Planning*, i.e., a sensor-based, real-time model that generates and periodically updates the path on–line in order to avoid both known and unforeseen obstacles [8];
- *Path Tracking*, i.e., a feedback-control model that is capable of driving a unicycle vehicle along the collision free path.

Given two points in the Cartesian Space, also referred to as *start* and *goal*, *Motion Planning* computes the most promising path to the goal in order to avoid perceived obstacles. If there is not an immediate danger of colliding with an obstacle, the start and the goal are simply connected through the shortest path: a straight line. Otherwise, during motion, range sensor data returned by sonars or by a laser scanner are used to periodically update the path to guarantee safe navigation. To achieve this, differently from other approaches, the distance from obstacles is neither used to build a local map, nor to deform the whole path as sometimes proposed in literature [17][18][19][20]. The general idea is that of *artificially* reducing or increasing the position error (i.e., the distance to the reference straight path measured by *Path Tracking*) through simple, real–time computations that consider only obstacles in the immediate surroundings, thus implicity defining a new path that is guaranteed not to collide with the obstacles themselves.

Path Tracking allows to regulate to zero a) the distance to the path, possibly taking into account the error purposely added to avoid collisions with obstacles, as well as b) the difference between the vehicle's orientation and the tangent to the curve, and it is proven to be *asymptotically stable*. The model is different from other models in literature [2] [3] [7] in the following two points:

- only the distance $D(x, y)$ between the vehicle and the path is measured and fed to the controller, whereas most approaches require to measure both the distance from the path *and* the difference between the desired and the actual orientation;
- even if the resulting path is not known a priori and possibly changes in run-time depending on *Motion Planning*, it is demonstrated that the resulting curve is an analytic function in Cartesian Space described by its implicit equation $y - E(x) = 0$, where $E(x)$ is the error which is artificially added in real-time by *Motion Planning* to take into account deviations from the actual path due to the presence of obstacles.

The system has some unique characteristics, both when considering the two components separately and as a whole.

Motion Planning requires very few computational resources, and for this reason it can be used in any kind of unknown or changing environment, even by very simple robots. This is mainly due to the fact that it reactively updates the path using only local sensor information (see [9][10][11][12][13][14][15]): reactive motion planning techniques are known to be computationally more efficient than global planning techniques, since the latter compute a complete path to the goal and require a global model of the environment (see [1] and the references therein). However, this is not the only reason. With respect to other path-deformation approaches [17][18][19][20], the proposed model needs even less computational resources, since it initially computes a straight path to the goal, but this path is never really "deformed" to take into account surrounding obstacles: in fact, this would require complex mathematical tools to represent the concept of a "deformed path", such as sampling the path at a sufficiently high resolution, and moving the resulting vector of sample points in the workspace to avoid intersections with obstacles. Instead, obstacle avoidance is achieved by simply considering the current position error (i.e., a single point in the path instead of a vector of points), and by adding/deleting a scalar quantity to/from such error according to a properly defined strategy.

The fact that *Path Tracking* does not require to measure the vehicle's orientation is very important, since the orientation is usually more affected by errors and more difficult to measure

A. Sgorbissa, A. Villa, A. Vargiu, R. Zaccaria are with DIST, Department of Communication, Computer and System Sciences, University of Genoa, Via Opera Pia 13, 16145, Genoa, Italy. Corresponding Author email:{antonio.sgorbissa}@unige.it.

with exteroceptor sensors. An example is wall following: the distance from the wall can be sensed through proximity sensors (i.e., sonar, laser scanners, etc.) by considering raw measurements alone (distance corresponds to the minimum sensed range), whereas measuring orientation requires more complex computations. Another example is GPS-based outdoor navigation: even if commercial devices return velocity (and hence heading information), this measurement is accurate only when velocity is high, and therefore it turns out not to be enough reliable for path tracking in a general case. Gyroscopes have the problem of temporal drift, and therefore they are not a reliable source of information in the long term. Notice that – with a differentially driven unicycle vehicle – even a small error in wheels encoders can produce a significant error in orientation when moving at low speed. For a deeper investigation, see also [21].

Section II describes general ideas; Section III considers the problem of avoiding a single in-path stationary obstacle modeled as an ellipse, and extends the previous case to N obstacles; Section V describes an asymptotically stable control law which drives a vehicle along the planned path; finally, Section VI shows experimental results. Conclusions follow.

II. Motion Planning - General Ideas

The following case is considered: a vehicle is following a generic straight path when it senses an obstacle on its path to the goal. The vehicle must be able to avoid the obstacle in order to safely reach the goal.

Consider a Cartesian's reference system: without loosing generality, it is always possible to choose the X-axis as lying along the vehicle path (a straight line). Under these conditions the vehicle path can be written as $y = 0$.

In order to avoid the sensed obstacle, one could be tempted to re-plan the whole path. However, a different approach can be pursued; in particular, it is possible to define an error function:

$$E = E(x) \tag{1}$$

The error in (1) represents, for each point along the $X-$axis, a safety distance from the reference straight path that allows to avoid the obstacle. Consider, for example, an obstacle modeled as a circle with radius R and center in $(x_o, y_o = 0)$, i.e., on the reference path (Figure 1). In this case the minimum error function $E(x)$ that guarantees to avoid the obstacle has the expression:

$$\begin{array}{ll} E(x) = 0 & x < x_o - R \quad or \quad x > x_o + R \\ E(x) = R\sin(\beta(x)) & x_o - R \leq x \leq x_o + R \end{array} \tag{2}$$

where x is the vehicle's position along the X-axis, and $\beta(x) = cos^{-1}((x - x_o)/R)$ is the angle between the $X-$axis and the straight line connecting the obstacle center $(x_o, 0)$ to the corresponding point along the obstacle's boundary $(x, E(x))$.

$E(x)$ has the following properties:

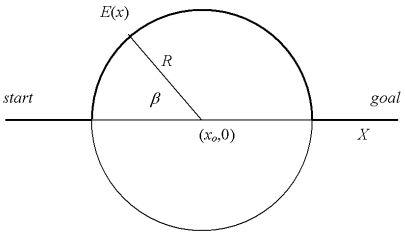

Fig. 1. The reference path $y = 0$, a circular obstacle with radius R, and the corresponding error function $E(x)$.

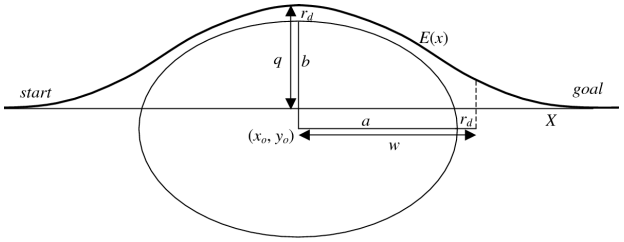

Fig. 2. The reference path $y = 0$, an elliptic obstacle with axes a and b, and the corresponding error function $E(x)$.

- when $x < x_o - R$ or $x > x_o + R$, $E(x)$ lies on the straight line;
- when $x_o - R \leq x \leq x_o + R$, $E(x)$ lies on the obstacle's boundary;
- $E(x)$ depends only on the current position of the vehicle and on surrounding obstacles, and can be computed in real-time through proximity sensors (e.g., a laser rangefinder or ultrasounds).
- $E(x)$ is a C^1 piecewise curve with an analytical expression.

The general idea is that, for each x along the initial path, *Motion Planner* calculates $E(x)$ on the basis of sensor data. This can be done by clustering data and computing the corresponding bounding circle, or even by considering each sensor reading as a separate obstacle, whose radius takes into account the vehicle dimensions plus a safety distance. Next, it ideally adds $E(x)$ to the original straight path, which practically corresponds to subtracting $E(x)$ from the positioning error in *Path Tracking*: the final effect is that, when the robot is moving along the obstacle's boundary, its distance to the reference straight path is exactly $D(x, y) = E(x)$, but the positioning error after subtracting $E(x)$ is artificially set to zero.

Remark: since $E(x)$ depends only on the current position of the vehicle and on surrounding obstacles, it is particularly suited to deal with moving or suddenly appearing obstacles: there is never a waste of computational resources for re-planning, since *Motion Planning* computes exclusively *what is needed here and now*, without making any hypotheses on what will happen in the close future.

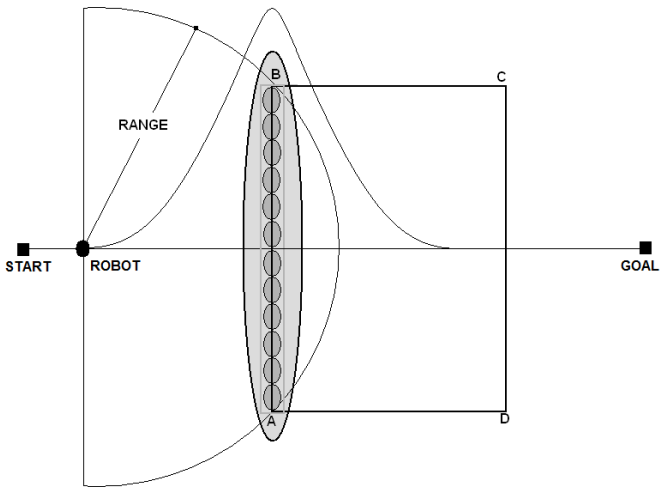

Fig. 3. The robot perceives the AB side of the obstacle.

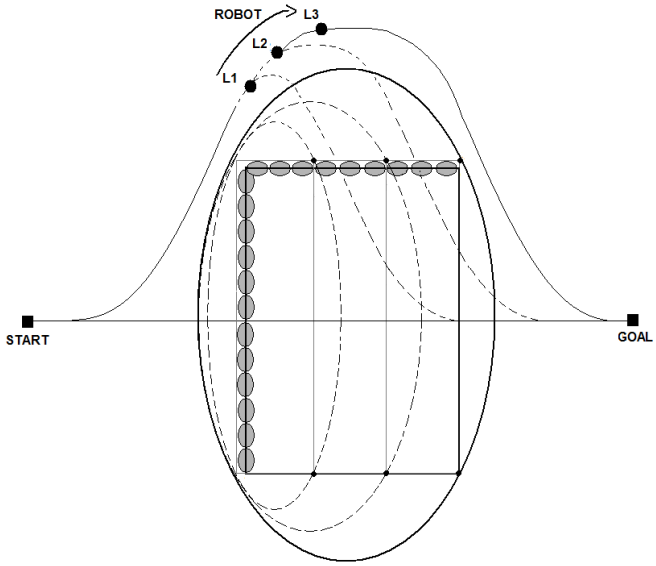

Fig. 4. As the robot moves forward, it periodically updates the ellipse center and $E(x)$.

III. The obstacle model - clustering

The expression of $E(x)$ in (1) has some limitations: it is only piecewise C^1 and it models obstacles as circles, which can turn out to be not very efficient especially in cluttered environments. To overcome these limitations, a slightly more complex model is proposed, in which obstacles are modeled as ellipses with center (x_o, y_o) in a Cartesian space (Figure 2):

$$\frac{(x-x_o)^2}{a^2} + \frac{(y-y_o)^2}{b^2} = 1 \tag{3}$$

Without loosing generality, the ellipse main axes are assumed to lie along the $X-$ and $Y-$axis of the reference frame, with a and b corresponding to their respective lengths. The center (x_o, y_o) can be everywhere in the XY plane. It is worth noticing that typical proximity sensors (e.g., ultrasonic sensors or lasercanners) are able to measure the distance from the surface of the obstacle rather than from its centre. To understand how the coordinates of the obstacle center are inferred starting from proximity data, consider Figure 3.

An obstacle $ABCD$ with a rectangular shape intersects the straight line connecting the start to the goal. The robot, while approaching the obstacle, is able to perceive side AB of the obstacle through the measurements returned by a laserscanner. The raw laser measurements initially appear as a cloud of small obstacles distributed along the obstacle profile: in Figure 3 range measurements are shown as ellipses, whose number is obviously much higher in the real case (laser scanners have an angular resolution of at least 2 raw measurements per degree). Consequently, it is necessary to recursively cluster raw measurements in order to interpret them as belonging to the same obstacle. To this purpose, couples of neighbouring obstacles are recursively considered, which allows to finally obtain – in case that clustering conditions are met – a single obstacle whose dimension is properly set to allow safe and smooth obstacle avoidance. In particular, a couple of obstacles are clustered if the following conditions hold:

- the distance between two neighboring obstacles along the Y-axis is not sufficient to allow the vehicle passing between them;
- the distance between two neighboring obstacles along the X-axis is not sufficient to allow the vehicle moving back to the original path, once the first obstacle has been avoided.

In Figure 3 the first rule is recursively applied, with the final result that all smaller obstacles (corresponding to raw laser measurements) are clustered to form a bigger ellipse that contains the AB side of the obstacle. In order to avoid it, the robot computes the center of the ellipse as well as the corresponding error $E(x)$, and starts moving along the corresponding path. As the robot moves forward, the BC side of the obstacle becomes visible as well: laser scannner returns new measurments that, once again, can be represented as small obstacles distributed along BC (Figure 4). The second rule above is recursively applied and the smaller obstacles along BC are recursively clustered as well, with the final result that the ellipse that initially enclosed AB is enlarged in such a way as to contain BC as well. The center of this ellipse and the error $E(x)$ are updated correspondingly. Finally, when the vehicle reaches a position from where it can observe the entire obstacle side BC, the resulting ellipse completely surrounds the obstacle $ABCD$, thus allowing smooth and safe obstacle avoidance.

IV. Motion Planning - Computing the error function

To guarantee that the robot stay close to the obstacle's boundary while avoiding it, it seems reasonable to model the error function $E = E(x)$ as a Gaussian function, which "bell-shaped profile" (see Figure 2) can be written in the form:

$$E(x) = q \cdot e^{-\frac{(x-x_c)^2}{w^2}} \tag{4}$$

To this purpose, it is necessary to choose the Gaussian parameters q and w properly, in such a way that $E(x)$ adheres to the ellipse boundary as close as possible, while taking safety distance into account.

In order to compute q, consider that it represents the maximum function amplitude: i.e., $x - x_o = 0 \Rightarrow E(x) = q$. Since, when $x - x_o = 0$, the vehicle must be far from the obstacle center at least as much as the length b of the corresponding axis, and the distance from the reference straight path is equal to y_o, it is possible to set

$$\begin{array}{ll} q = y_o + b + s_d \cdot r_d & (-b - r_d \leq y_o \leq b + r_d) \\ q = 0 & (y_o < -b - r_d \quad or \quad y_o > b + r_d) \end{array} \tag{5}$$

where r_d is a safety distance that takes into account the vehicle dimension, and s_d is a gain that allows to increase/decrease the relative importance of r_d in (5).

Equation 5 assumes that the robot tries to avoid an obstacle only when the latter intersects the reference straight path, i.e., when $(-b - r_d \leq y_o \leq b + r_d)$; moreover, this is always done by increasing y, i.e., the robot always avoids the obstacle by turning on its left. However, it would be more efficient to properly choose the sign of q in such a way that it can either be negative or positive, depending on which is the shortest path to avoid the obstacle. In fact, if $-b - r_d \leq y_o \leq 0$, the most promising path to avoid the obstacle can be found by increasing y; if $0 < y_o \leq b + r_d$, the opposite is true. To achieve this, it is necessary to introduce a function, i.e., an expression of the sign of y:

$$sign(y) = \begin{cases} 1 \text{ when } y \geq 0 \\ -1 \text{ when } y < 0 \end{cases} \tag{6}$$

The first line in (5) becomes:

$$q = y_o - sign(y_o) \cdot (b + s_d \cdot r_d) \qquad (-b - r_d \leq y_o \leq b + r_d) \tag{7}$$

In order to compute w, consider that the robot must return to the reference straight path $y = 0$ after having avoided the obstacle, i.e., when $x > x_o + a$. Since the Gaussian in (10) never equals zero, the concept of "lying on the reference straight path" should be meant as an approximation, which can be expressed as a function of the robot dimension r_d. That is, after the robot has avoided the obstacle, it must hold:

$$|q| \cdot e^{-\frac{(x-x_o)^2}{w^2}} \leq 0.1 r_d \tag{8}$$

By solving (8) for x it holds:

$$\begin{array}{l} x \leq x_o - w\sqrt{\ln \frac{|q|}{0.1 r_d}} \\ x \geq x_o + w\sqrt{\ln \frac{|q|}{0.1 r_d}} \end{array} \tag{9}$$

Notice that the line $y = 0.1 r_d$ should intersect the Gaussian, in fact the following condition must be satisfied:

$$\ln \frac{|q|}{0.1 r_d} \geq 0 \Rightarrow |q| \geq 0.1 r_d \tag{10}$$

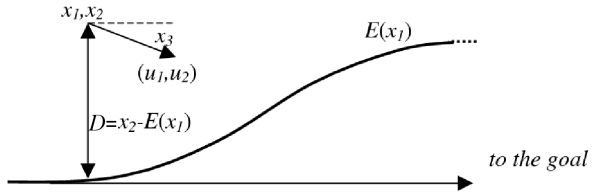

Fig. 5. Path Tracking by controlling the rotational speed u_2.

Since the vehicle returns to the reference path after having avoided the obstacle, only the second set of solutions is significant. It is now possible to compute a proper value for w such that:

$$x_o + w\sqrt{\ln \frac{|q|}{0.1 r_d}} > x_o + a \tag{11}$$

By choosing $w = a$, (11) implies that

$$\sqrt{\ln \frac{|q|}{0.1 r_d}} > 1 \tag{12}$$

which holds whenever

$$|q| > e \cdot 0.1 r_d \approx 0.27 r_d \tag{13}$$

The relation in (13) is always satisfied for q, given that the gain s_d in (10) is properly set. For example, if $s_d = 1.3$, (10) returns $q = 0.3 r_d$ (or $q = -0.3 r_d$) even in the worst case when $y_o = -b - r_d$ (or $y_o = b + r_d$), i.e., when the obstacle is "almost tangent" to the reference path. To adopt a more conservative approach, it is finally set $w = a + r_d$.

When more obstacles are detected by sensors which cannot be clustered according to the conditions above, each obstacle is modeled separately as an ellipse. In general, calculating $E(x)$ does not present more difficulties than the simpler case in which only one obstacle is present: each single obstacle in the workspace is considered as if it were the only one, and a positioning error is consequently computed according to (4). Next, all contributes are summed up. Formally, it can be written:

$$E(x) = \sum_{k=1}^{N} q_k \cdot e^{-\frac{(x-x_{o,k})^2}{(a_k + r_d)^2}} \tag{14}$$

where N is the total number of the surrounding obstacles detected by sensors. As before, the path $E(x)$ that takes into account all obstacles is not planned a priori: instead, it is computed only on the basis of the current position of the vehicle and on surrounding obstacles, thus being able to deal efficiently with obstacles that move, or which suddenly appear and disappear from the field of view.

V. Path Tracking

In order to show how *Path Tracking* works, it is useful to define a state vector $\mathbf{x}$ (Figure 5):

$$\mathbf{x}^T = [\ x_1 \quad x_2 \quad x_3\ ] = [\ x \quad y \quad \vartheta\ ] \tag{15}$$

As usual, the unicycle kinematics can be described through the following state equations:

$$\begin{aligned} \dot{x_1} &= u_1 \cos x_3 \\ \dot{x_2} &= u_1 \sin x_3 \\ \dot{x_3} &= u_2 \end{aligned} \tag{16}$$

where inputs u_1 and u_2 are – respectively – the translational and the rotational velocities.

The control model is aimed at regulating to zero both the distance $D(x_1, x_2)$ between the vehicle's position and the path and the difference between the vehicle's orientation and the tangent to the path. In particular, it is necessary to demonstrate the asymptotic stability of the system during obstacle avoidance, i.e., when *Path Tracking* drives the vehicle along a path defined by the function:

$$x_2 = E(x_1) \tag{17}$$

Path Tracking is defined here as the problem of minimizing the distance $D(x_1, x_2)$ along the Y-axis between the vehicle's position (x_1, x_2) and the curve defined in explicit form in (17), that is:

$$D(x_1, x_2) = x_2 - E(x_1) \tag{18}$$

Strictly speaking, $D(x_1, x_2)$ is not the Euclidean distance between (x_1, x_2) and the path, since it is always computed along the Y-axis and hence it does not correspond to the shortest distance. However, it still has some good properties which make it appropriate for our purpose:

- $D(x_1, x_2)$ is a scalar field.
- $D(x_1, x_2) = 0$ when (x_1, x_2) lies on the curve; $D(x_1, x_2)$ locally increases/decreases monotonically depending on which side of the plane (x_1, x_2) is located with respect to the curve.

From (18) it derives, by omitting to write the dependence of D on (x_1, x_2):

$$\begin{aligned} D_{x_1} &= \frac{\partial D}{\partial x_1} = \frac{\partial E(x_1)}{\partial x_1} \\ D_{x_2} &= \frac{\partial D}{\partial x_2} = 1 \end{aligned} \tag{19}$$

Where D_{x_1} and D_{x_2} are respectively the partial derivatives of D with respect to x_1 and x_2.

It can be demonstrated that, in order to guarantee stability, it is possible to set:

$$\begin{aligned} u_1 &= U_1 \\ u_2 &= K \|\nabla D\| (-D - \tfrac{d}{dt} D) + \tfrac{d}{dt} \tan^{-1} \left(\tfrac{-D_{x_1}}{D_{x_2}} \right) \end{aligned} \tag{20}$$

The rotational velocity u_2 is given by a term proportional to the distance plus a term which depends on the curvature of the isocline in the current position.

The underlying idea is simple. Consider that, according to (18), D is equal to the distance between the current robot's position and the path that allows obstacle avoidance. The derivative of D with respect to time is referred to as *approaching velocity*: in Figure 5, the *approaching velocity* is negative, since the distance between the robot and the path decreases in time. Notice that the *actual approaching velocity* $\frac{d}{dt} D$ can be increased by controlling x_3, which – on its turn – requires to operate on the rotational speed u_2. In particular (20) sets the rotational speed u_2 as proportional to the difference between a *reference approaching velocity* $\dot{x}_{ref} = -D$ (i.e., computed as the inverse of the distance) and the *actual approaching velocity* $\frac{d}{dt} D$. The *reference approaching velocity* has the following properties:

- when $x_2 = E(x)$, $\dot{x}_{ref} = 0$ as well (the robot lies on the path);
- when $x_2 > E(x)$, $\dot{x}_{ref} = -D$ is negative (heading downward in Figure 5);
- when $x_2 < E(x)$, $\dot{x}_{ref} = -D$ is positive (heading upward in Figure 5).

The translational speed u_1 is a free variable and can have a generic profile $U_1(t)$ (given that it satisfies kinematics and dynamics constraints). In the following it is assumed that $U_1(t) = U_1$ is constant.

The whole system can be expressed as:

$$\begin{aligned} \dot{x_1} &= U_1 \cos x_3 \\ \dot{x_2} &= U_1 \sin x_3 \\ \dot{x_3} &= K \|\nabla D\| (-D - \tfrac{d}{dt} D) + \tfrac{d}{dt} \tan^{-1} \left(\tfrac{-D_{x_1}}{D_{x_2}} \right) \end{aligned} \tag{21}$$

In [21] it is formally demonstrated that, when adopting the control law in (20), both the distance error $x_2 - E(x_2)$ and the orientation error $x_3 - \tan^{-1} \left(\frac{-D_{x_1}}{D_{x_2}} \right)$ tends asymptotically to zero.

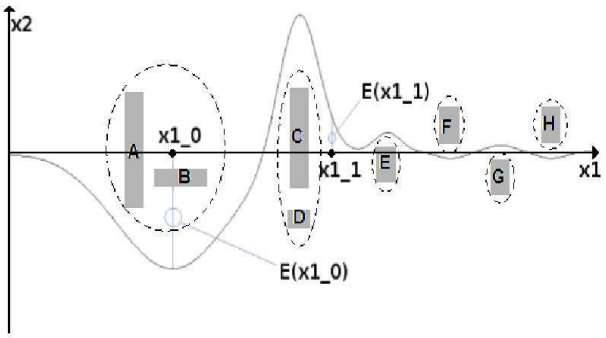

Fig. 6. Simulation run 1

VI. Experimental results

Many experiments have been performed in simulation in the SimuLink environment, both with moving and stationary obstacles. Moreover, preliminary experiments have been performed with real robots.

As an example, Figures 6 and 7 show two simulation runs. In Figure 6, the couple A and B is clustered, and the same happens to C and D. The clustering algorithm does not cluster E, F, G and H because they are enough far from each other to allow the vehicle to safely pass through. In Figure 7, the first three obstacles are clustered, whereas the latter two are not. In all the experiments, simulated sensors have a very limited sensing range and cannot perceive obstacles that are occluded by other obstacles; in spite of this, smooth and safe obstacle avoidance is guaranteed in all situations.

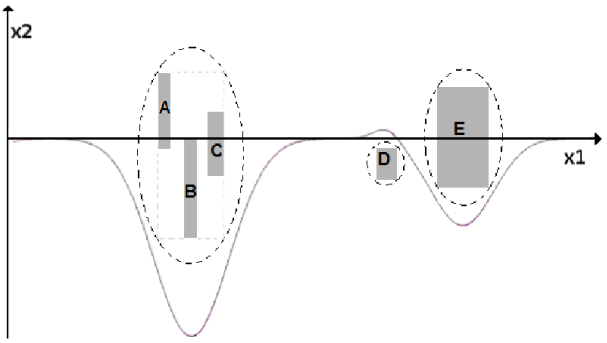

Fig. 7. Simulation run 2

In Table I, simulated results are summarized. Obstacles are randomly distributed in the environment, according to a parameter o_d that determines "obstacle density", i.e., how much cluttered with obstacles the environment is. In particular, 20 simulations have been ran for each density value, i.e., *Low density* ($\approx$ 3 obstacles), *Medium density* ($\approx$ 5 obstacles), and *High density* ($\approx$ 10 obstacles). In all simulations the vehicle has to follow a reference straight path for about $50m$. Realistic noise is added to motion control, by adding white noise with zero mean and a 2% standard deviation both to the left and the right wheel velocity, i.e., ω_l and ω_r. Moreover, a systematic error of 5% has been added to ω_r, to simulate a noisy estimate of the vehicle geometric parameters. The second column of Table I reports the average error $\overline{D}$ between the ideal path and the real path of the robot, whereas the third column reports the correspondent standard deviation σ_D. It can be easily notices that – even if experiments in the real-world are necessary before making strong claims about the validity of the approach –, simulated results validate the theoretical assumptions made throughout the article.

VII. Conclusions

Preliminary experiments with moving obstacles have been performed as well, showing that – in its general ideas – the approach can be immediately applied to this more complex situation. However, in the case of moving obstacles, the system is not always able to generate collision-free path even when the clustering algorithm is active. A preliminary investigation has been conducted to increase the efficiency of the algorithm to deal with moving obstacles: notice however that – in general – it does not seem possible to guarantee smooth obstacle avoidance in *all* situations, if obstacles are able to move *fast enough*, and they behave *as if* they did not want the robot to avoid them (which can be the case of a human that really wants to stop the robot). Should this happen, it seems reasonable to accept that the robot must simply stop, and possibly starts a recovery procedure that can involve planning, but also pronouncing vocal messages to kindly ask surrounding people to move on. These issues are currently investigated.

TABLE I

Experiments with added Gaussian noise.

o_d	$\overline{D}$	σ_D
Low	0.0104	5.6505e-005
Medium	0.0092	5.0277e-005
High	0.0063	3.5898e-005

References

[1] Steven M. Lavalle, Planning Algorithms, Cambridge Univ Press, 2006

[2] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino. Closed Loop Steering of Unicyle-Like Vehicles via Lyapunov Techniques. IEEE Robotics and Automation Magazine, 1995.

[3] Giovanni Indiveri and Andreas Nüuchter and Kai Lingemann. High Speed Differential Drive Mobile Robot Path Following Control With Bounded Wheel Speed Commands. 2007 IEEE International Conference on Robotics and Automation Roma, Italy, 10-14 April 2007

[4] Samson, C. and Ait-Abderrahim, K., Mobile Robot Control Part 1: Feedback Control of A Non-Holonomic Mobile Robots, Technical Report No. 1281, INRIA, Sophia-Antipolis, France, June 1991

[5] C. Canudas de Wit, H. Khennoul, C. Samson, and O. J. Sordalen, Nonlinear control design for mobile robots, in Recent Trends in Mobile Robots, ser. Robotics and Automated Systems, Y. F. Zheng, Ed. World Scientific, 1993, ch. 5, pp. 121156.

[6] Z. P. Jiang and H. Nijmeijer, A recursive Technique for Tracking Control of Nonholonomic Systems in Chained Form, IEEE Trans. on Robotics and Automation, Vol 44, No 2, 1999, pp. 265-279

[7] Marvin K. Bugeja and Simon G. Fabri Dual Adaptive Control for Trajectory Tracking of Mobile Robots, 2007 IEEE International Conference on Robotics and Automation Roma, Italy, 10-14 April 2007

[8] Lionel Lapierre, Rene Zapata and Pascal Lepinay, Simultaneous Path Following and Obstacle Avoidance Control of a Unicycle-type Robot, 2007 IEEE International Conference on Robotics and Automation Roma, Italy, 10-14 April 2007

[9] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, Int. Journal of Robotics Research, vol. 5, no. 1, 1986.

[10] J. Borenstein and Y. Korem, The vector field histogram – fast obstacle avoidance for mobile robts, IEEE Trans. Robotics and Automation, vol. 7, no. 3, pp. 278–288, June 1991.

[11] D. Fox, W. Burgard, and S. Thrun, The dynamic window approach to collision avoid- ance, IEEE Robotics and Automation Magazine, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[12] N. Y. Ko and R. Simmons, The lane-curvature method for local obstacle avoidance, in Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, Victoria, BC (CA), Oct. 1998, pp. 1615–1621.

[13] O. Brock and O. Khatib, High-speed navigation using the global dynamic window approach, in Proc. of the IEEE Int. Conf. on Robotics and Automation, Detroit, MI (US), May 1999, pp. 341–346.

[14] P. Fiorini and Z. Shiller, Motion planning in dynamic environments using velocity obstacles, Int. Journal of Robotics Research, vol. 17, no. 7, pp. 760–772, July 1998.

[15] J. Minguez and L. Montano, Nearness diagram (ND) navigation: collision avoidance in troublesome scenarios, IEEE Trans. on Robotics and Automation, vol. 20, no. 1, pp. 45–59, Feb. 2004.

[16] Hanna Kurniawati and Thierry Fraichard, "From Path to Trajectory Deformation", August 28, 2007

[17] M. Khatib, H. Jaouni, R. Chatila, and J.-P. Laumond, Dynamic path modification for car-like nonholonomic mobile robots, in Proc. of the IEEE Int. Conf. on Robotics and Automation, Albuquerque, NM (US), Apr. 1997, pp. 2920–2925.

[18] O. Brock and O. Khatib, Elastic strips: a framework for motion generation in human environments, Int. Journal of Robotics Research, vol. 21, no. 12, pp. 1031–1052, Dec. 2002.

[19] F. Lamiraux, D. Bonnafous, and O. Lefebvre, Reactive path deformation for nonholo- nomic mobile robots, IEEE Trans. on Robotics and Automation, vol. 20, no. 6, pp. 967–977, Dec. 2004.

[20] Y. Yang and O. Brock, Elastic roadmaps: Globally task-consistent motion for au- tonomous mobile manipulation, in Proc. of the Robotics: Science and Systems, Philadelphia, USA, August 2006.

[21] A. Sgorbissa and R. Zaccaria, A Minimalist Feedback Control for Path Tracking in Cartesian Space, 2009 IEEE/RSJ International Conference on Intelligent RObots and Systems, October 11 – 15, 2009, St. Louis, MO, USA