Autonomous Robots 19, 159–171, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Navigation Among Moving Obstacles Using the *NLVO*: Principles and Applications to Intelligent Vehicles

FRÉDÉRIC LARGE AND CHRISTIAN LAUGIER
INRIA Inria Rhône-Alpes, Grenoble, France
christian.laugier@inria.fr; http://www.inria.fr

ZVI SHILLER
College of Judea and Samaria, Ariel, Israel
shiller@yosh.ac.il

Abstract. Vehicle navigation in dynamic environments is a challenging task, especially when the motion of the obstacles populating the environment is unknown beforehand and is updated at runtime. Traditional motion planning approaches are too slow to be applied in real-time to this problem, whereas reactive navigation methods have generally a too short look-ahead horizon. Recently, iterative planning has emerged as a promising approach, however, it does not explicitly take into account the movements of the obstacles.

This paper presents a real-time motion planning approach, based on the concept of the *Non-Linear V-Obstacle* (*NLVO*) (Shiller et al., 2001). Given a predicted environment, the *NLVO* models the set of velocities which lead to collisions with static and moving obstacles, and an estimation of the times-to-collision. At each controller iteration, an iterative A* motion planner evaluates the potential moves of the robot, based on the computed *NLVO* and the traveling time. Previous search results are reused to both minimize computation and maintain the global coherence of the solutions.

We first review the concept of the *NLVO*, and then present the iterative planner. The planner is then applied to vehicle navigation and demonstrated in a complex traffic scenario.

Keywords: mobile robot, obstacle avoidance, iterative motion planning, moving obstacles, velocity-obstacle

1. Introduction

Autonomous vehicle navigation among stationary obstacles can now be considered as solved. This is not the case for environments containing moving obstacles whose future behavior is unknown: Their future trajectories should be taken into account to anticipate collisions and to compute a safe trajectory accordingly. When this information is not available beforehand and has to be estimated at runtime, wrong predictions may be done and the safety of the vehicle depends on its ability to react fast to updates. The available computation time depends on the dynamics of the environment and on the controllability of the robot.

Existing autonomous navigation approaches are classically split between motion planning methods for which a complete trajectory to the goal is computed once, e.g. (Erdmann and Lozano-Perez, 1986; Fujimura and Samet, 1989), and reactive methods, for which only the next move is computed, e.g. (Fox et al., 1997; Ulrich and Borenstein, 1998; Ko and Simmons, 1998; Minguez and Montano, 2000). Planning methods are too slow, whereas reactive ones have too little look-ahead. More recent methods, e.g. (Fraichard, 1999; Brock and Khatib, 1999; Hsu et al., 2000; Brock and Khatib, 2000; Minguez et al., 2002; Frazzoli et al., 2002; Stachniss and Burgard, 2002), called iterative motion planners, tend to extend the reactive methods with planning techniques. Instead of computing only the next move, several steps are computed, depending on the time available. Different possibilities are explored in a search-tree, and a partial

trajectory is incrementally built. The process can be interrupted at any time to keep the vehicle reactive, while the trajectory returned is the best among the ones explored in the allocated time.

This approach is the most promising, however, it requires an important condition not yet satisfied in current methods: The environment and its dynamics must be represented in a more efficient way (e.g. in spaces of lower dimension to reduce the response time).

This paper presents an iterative planner that uses the *Non-Linear $\mathcal{V}$-Obstacle* (Shiller et al., 2001) to estimate efficiently the safety of the vehicle's motion in a predicted environment. This process is iteratively repeated to incrementally build a search tree, until a complete trajectory to the goal is found, or until the available computing time is exhausted. The tree is continuously updated to reflect the environmental changes.

We first review the concept of *Non-Linear $\mathcal{V}$-Obstacle* in Section 2 and present a few extensions to be used in intelligent vehicle applications. Our iterative motion planner is detailed in Section 3. Experimental results obtained on our simulator are interpreted for a realistic traffic example in Section 4.

2. The Nonlinear V-Obstacle *NLVO*

2.1. *Principles of NLVO*

To understand the *NLVO* concept, let us first consider a simple case under several assumptions. The derived definitions will be later extended to handle more realistic problems.

2.1.1. Hypothesis and Notations. Let us consider a single circular robot $\mathcal{A}$ and a single circular obstacle $\mathcal{B}$ both moving in the plane noted as $\mathcal{W}$. $\mathcal{A}(t)$ and $\mathcal{B}(t)$ are the subsets of $\mathcal{W}$ occupied by $\mathcal{A}$ and $\mathcal{B}$ at instant t. $c_A(t)$ and $c_B(t)$ are the configurations of $\mathcal{A}$ and $\mathcal{B}$ at instant t. For convenience, $\mathcal{A}(t)$ is reduced to its center $c_A(t)$ and $\mathcal{B}(t)$ is grown by the radius of $\mathcal{A}(t)$ and is noted $\mathcal{CB}(t)$. $\overrightarrow{v_A}(t)$ and $\overrightarrow{v_B}(t)$ are the instantaneous linear velocities of $\mathcal{A}$ and $\mathcal{B}$ at instant t. They are represented by two vectors, attached respectively to $c_A(t)$ and $c_B(t)$, in the velocity space (Burke, 1985) of the robot noted as $\mathcal{V}$.

The trajectories of $\mathcal{A}$ and $\mathcal{B}$ are considered on a bounded time interval $[t_0, TH]$, where TH is known as the *time horizon* (Fiorini and Shiller, 1998). The trajectory of $\mathcal{A}$ is considered as linear and constant on $[t_0, TH]$. The trajectory of $\mathcal{B}$ is arbitary. However, $c_B(t)$ and $\overrightarrow{v_B}(t)$ are assumed to be known or predictable on $[t_0, TH]$ (see Large et al., 2004 for an example of a prediction method and its integration with the *NLVO* concept).

2.1.2. Definitions. The concept of *$\mathcal{V}$-Obstacle* (noted as *VO*) has first been introduced in Fiorini and Shiller (1993, 1998), as *the set of all linear constant velocities of a robot that induce a collision with an obstacle on a bounded period of time.*

$$VO = \bigcup \{\overrightarrow{v_A} \in \mathcal{V} \mid \exists t \in [t_0, TH],\ \mathcal{A}(t) \cap \mathcal{B}(t) \neq \emptyset\} \tag{1}$$

A *temporal element* of a *VO* is the set of all velocities that cause collision between the robot and the obstacle at time t. It is noted as $VO(t)$:

$$VO(t) = \bigcup \{\overrightarrow{v_A} \in \mathcal{V} \mid \mathcal{A}(t) \cap \mathcal{B}(t) \neq \emptyset\} \tag{2}$$

A *VO* is the union of all its temporal elements for $t \in [t_0, TH]$:

$$VO = \bigcup_{t=t_0}^{TH} VO(t) \tag{3}$$

A *$\mathcal{V}$-Obstacle* is called *linear* (*LVO*) if the obstacle is static or if its trajectory is constant and linear. It becomes a *non-linear $\mathcal{V}$-Obstacle* (*NLVO*) if the trajectory of the obstacle is arbitrary (Shiller et al., 2001). Obviously, the *NLVO* accounts for general as well as for linear trajectories.

2.1.3. Construction of the NLVO. Let us first consider the simple case of a *LVO*. From a geometrical approach (Fiorini and Shiller, 1998), the shape of a *LVO* defined on $[t_0, TH]$ can be seen in $\mathcal{V}$ as a truncated cone delimited by the tangents to $\mathcal{CB}(t_0)$ issued from $c_A(t_0)$, and translated by $\overrightarrow{v_B}$ (Figs. 1.1 and 1.2). The truncated part is the apex of the cone. It corresponds to the velocities that induce a collision after TH (Fig. 1.3).

The explanation given in Fiorini and Shiller (1998) is repeated here. Let $\overrightarrow{v_{A/B}}$ be the velocity of $\mathcal{A}$ relative to $\mathcal{B}$. We can easily build the set of the relative velocities $\overrightarrow{v_{A/B}}$ such that $\mathcal{A}$ and $\mathcal{B}$ will collide at a future time. It is the cone delimited by the tangents to $\mathcal{CB}(t_0)$ issued from $c_A(t_0)$. We call it CC as *collision cone*. By definition, we know that $\overrightarrow{v_A} = \overrightarrow{v_{A/B}} + \overrightarrow{v_B}$. Hence, the set of the absolute velocities of $\mathcal{A}$ leading to a future collision is the translation of CC by the vector $\overrightarrow{v_B}$. Now, the

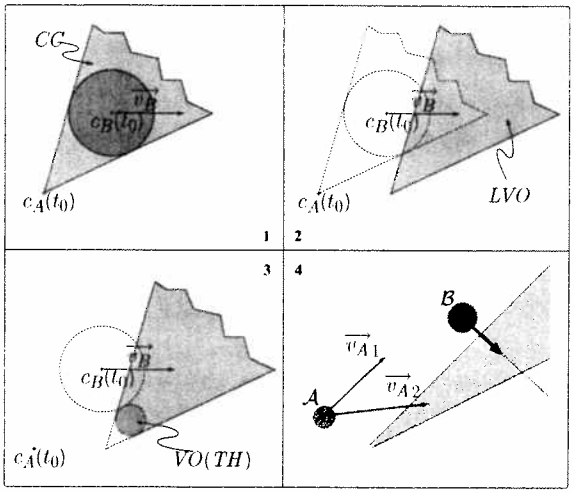

Figure 1. Geometrical construction of LVO in $\mathcal{V}$. 1. Construction of *CC* (*CC* represents the set of all the relative velocities of $\mathcal{A}$ related to $\mathcal{B}$ that induce a collision in a future time. It can be seen as the velocities of $\mathcal{A}$ that would induce a collision with $\mathcal{B}$ if $\mathcal{B}$ would not move. Hence, it can be easily built from $c_A(t_0)$ and $\mathcal{CB}(t_0)$), 2. Translation of *CC* by $\overrightarrow{v_B}$ to construct *LVO*, 3. Suppression of the extremity of the cone according to *TH* (computing *VO*(*TH*) allows to delimitate the "end" of the cone. The velocities of the cone located between the appex (included) and the border of *VO*(*TH*) (excluded) are dropped), 4. Example of a *LVO* computed in $\mathcal{V}$. (the construction of the *LVO* is independant of the current velocity $\overrightarrow{v_A}$ of $\mathcal{A}$). The *LVO* is used to evaluate the safety of the potential future velocities of $\mathcal{A}$. For example, the velocity $\overrightarrow{v_{A1}}$ is safe, while $\overrightarrow{v_{A2}}$ will induce a collision with the obstacle $\mathcal{B}$ at a time $t \in [t_0, TH]$ (the vector representing $\overrightarrow{v_{A2}}$ ends inside the *LVO*).

construction is equivalent to the union of all the $VO(t)$ for $t \in [t_0, \infty[$. We want to bound this range to $[t_0, TH]$ for two main reasons:

- In an enclosed area, any nonzero velocity of the robot would induce collision at some future time.
- The behaviour of the obstacles is unknown. It would be unrealistic to make any assumption on this behavior on an unbounded period of time.

We can observe that slower velocities, represented by shorter vectors issued from $c_A(t_0)$, lead to collisions in a further time. In other words, the velocities of the cone located between the apex and *VO*(*TH*) will induce a collision in $]TH, \infty[$: Thus, they have to be removed.

An analytical expression of the borders of the *LVO* was derived in Shiller et al. (2001): $VO(t)$ is the image of $\mathcal{CB}(t)$ in the scale drawing of ratio $1/t$ and of center $c_A(t_0)$ The velocities lying on the border of the *LVO* and inducing a collision at instant t, are the tangency

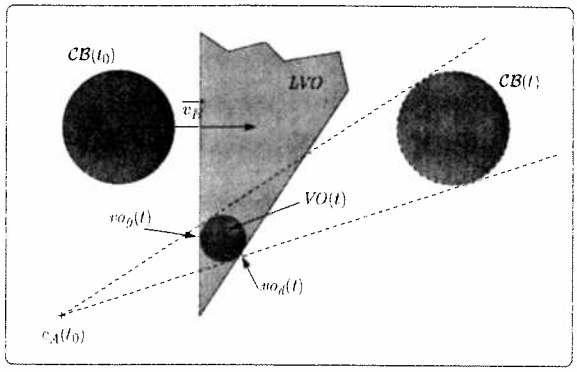

Figure 2. Construction of $vo_g(t)$ and $vo_d(t)$ for a *LVO*. $VO(t)$ is built as the image of $\mathcal{CB}(t)$ by the scaling transformation of center $c_A(t_0)$ and of ratio $1/t$. The translated point of $c_A(t_0)$ by $\overrightarrow{v_B}$ (t) is built. Two lines issued from this point and tangent to $VO(t)$ can be constructed. The tangency points are $vo_g(t)$ and $vo_d(t)$.

points noted as $vo_g(t)$ and $vo_d(t)$ between $VO(t)$ and the translation of $c_A(t_0)$ by $\overrightarrow{v_B}$ (Fig. 2). $VO(t)$ is computed as the image of $\mathcal{CB}(t)$ in the scale drawing of ratio $1/t$ and of center $c_A(t_0)$. The expression of $vo_g(t)$ and $vo_d(t)$ is the expression of the borders of the *LVO* parameterized by time.

The construction of a generic *NLVO* proposed in Shiller et al. (2001) extends the previous construction of a *LVO* and is based on the following property: the instantaneous linear velocity $\overrightarrow{v_B}$ (t) of $\mathcal{B}$ at time t along an arbitrary trajectory, is a first order approximation of this trajectory at time t. In a similar way, the *LVO* built from $\mathcal{B}(t)$ and $\overrightarrow{v_B}$ (t) can be seen as the first order approximation of a *NLVO* at time t.

The method consists of 3 steps, applied iteratively for discretized values of t in the time interval $[t_0, TH]$ with a time step dt (Fig. 3):

- consider $\overrightarrow{v_B}$ (t) to be constant on $[t_0, t]$ and estimate $\mathcal{CB}(t_0)$ (noted as $\mathcal{CB}_\circ$) from $\mathcal{CB}(t)$ under this assumption;
- compute the *LVO* noted as $LVO_\circ$ associated with $\mathcal{CB}_\circ$;
- compute the two tangency points $vo_g(t)$ and $vo_d(t)$ from the temporal element $VO_\circ(t)$ of $LVO_\circ$.

2.1.4. Time to Collision. Using the previous definition of *NLVO*, it is straightforward to say whether a given velocity will induce a collision in the future or will be safe. However, this definition gives no information about the time duration which is elapsed before the robot $\mathcal{A}$ collides with an obsctacle, while moving at a given constant velocity $\overrightarrow{v_A}$. This duration is called the

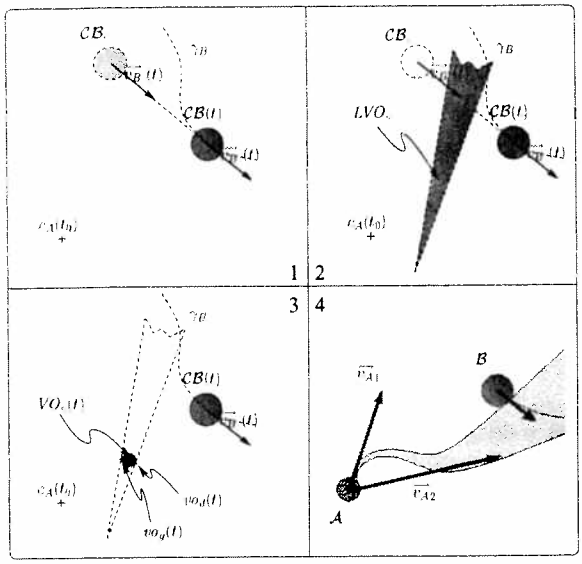

Figure 3. Construction of NLVO. (1) Construction of $\mathcal{CB}_\circ$ ($\mathcal{CB}_\circ = \mathcal{CB}(t) - t \cdot \overrightarrow{v_B}\ (t)$), (2) Construction of $LVO_\circ$ (from $c_A(t_0)$ and $\mathcal{CB}_\circ$) (3) Construction of $vo_g(t)$ and $vo_d(t)$ (in $LVO_\circ$) (4) Example of a *NLVO* computed in $\mathcal{V}$ (as seen previously, $\overrightarrow{v_{A1}}$ is safe, while $\overrightarrow{v_{A2}}$ will induce a collision with $\mathcal{B}$ at a time $t \in [t_0, TH]$).

time to collision. It can be obtained from the previous construction for the velocities lying on the border of the *NLVO*. In Large (2003), we proposed to also compute this time for the velocities located "inside" the NLVO.

To do so, the velocity space $\mathcal{V}$ is extended by a time dimension, corresponding to the time to collision. The new space is noted $\mathcal{V} \times \mathcal{T}$. Hence, a point in $\mathcal{V} \times \mathcal{T}$ corresponds to a linear and constant velocity of the robot, and the associated time to collision. The idea is to extend the construction of a *NLVO* in the space $\mathcal{V}$, to the space $\mathcal{V} \times \mathcal{T}$: A temporal element $VO(t)$ is computed for several discretized values of $t \in [t_0, TH]$. Its two tangent points $vo_g(t)$ and $vo_d(t)$ split its circumference into two arcs of a circle. The velocities that induce a collision with $\mathcal{B}_i$ at time t lie on the small arc of circle. Hence, the set of velocities with time to collision t lies on this small arc.

The construction of a *NLVO* in $\mathcal{V} \times \mathcal{T}$ extends the one in $\mathcal{V}$ and is as follows (Fig. 4): for each computed $VO(t)$, the small arc delimited by $vo_g(t)$ and $vo_d(t)$ is discretised in $n + 1$ points, giving $P_0(t)$ to $P_n(t)$ (with $vo_g(t) \equiv P_0(t)$ and $vo_d(t) \equiv P_n(t)$). The transfert from $\mathcal{V}$ to $\mathcal{V} \times \mathcal{T}$ implies that the time to collision t is added as a third coordinate. The 4 adjacent points $P_i(t)$, $P_{i+1}(t)$, $P_i(t + dt)$ and $P_{i+1}(t + dt)$ constitute a trapezoidal face of (the polygonal approximation of) *NLVO* in $\mathcal{V} \times \mathcal{T}$ (of dimension 3). Each one can be

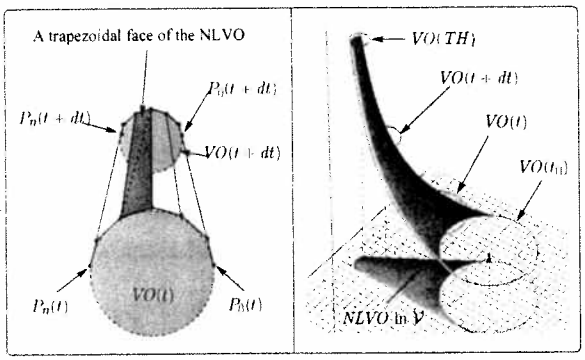

Figure 4. Polygonal approximation of a $\mathcal{V}$-Obstacle *in* $\mathcal{V} \times \mathcal{T}$. Left figure: The figure represents the construction of the trapezoidal faces that approximate a *NLVO* in $[t, t+dt]$. The points $P_i(t)$ are computed from the $VO(t)$. Each $P_i(t)$ has 3 coordinates: the 2 components of the corresponding linear velocity, and the time to collision associated with this velocity. $P_i(t)$ points are then grouped as described in the text to form quads or triangles as shown. Right figure: The previous triangles (or quads) are computed for all the discretized values of $t \in [t_0, TH]$ by steps of dt. They form the complete *NLVO* in $\mathcal{V} \times \mathcal{T}$. $VO(t_0)$, $VO(TH)$ and some intermediate $VO(t)$ have been added (circles) for a better understanding, as well as the projection of the *NLVO* in $\mathcal{V}$.

splitted into 2 triangles to enhance the performances (see *real-time issues* later in 4).

2.2. *Extensions of the Basic Problem*

In this section, the *NLVO* concept is extended to handle more complicated and realistic problems (see Large (2003) for details and extra extensions).

2.2.1. Multiple Obstacles. Considering now a world populated by several obstacles $\mathcal{B}_i$, the set of velocities that will induce a collision with any obstacle on $[t_0, TH]$ is the union of all the NLVO associated with each single obstacle. If a velocity induces a collision with several obstacles, its time to collision is the shortest duration the robot can travel at this velocity before the first collision occurs.

2.2.2. Non-Circular Robot. Approximating the robot shape by a single disk allows great simplifications of the calculations. However, a finer approximation using several disks is required in cluttered environments.

Two cases have to be distinguished, depending on the type of motion executed by the robot (pure translation or combination of translation and rotation).

In this paper, the considered robot is a car-like vehicle. Its trajectories are usually smooth; they can be approximized by a sequence of pure translations[1] over

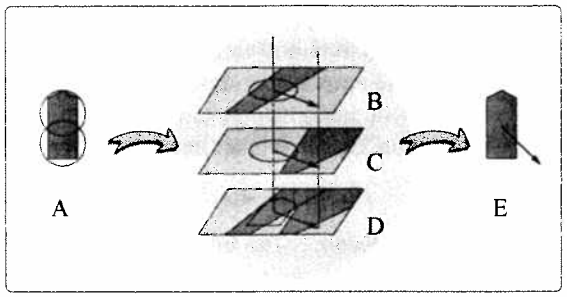

Figure 5. Approximating the NLVO associated with a non-circular obstacle in translation. The robot shape is approximated by two disks (A). The *NLVO* associated with the disks are computed (B and C) and their union is computed (D) to approximate the *NLVO* associated with the robot. A safe velocity can then be selected (E).

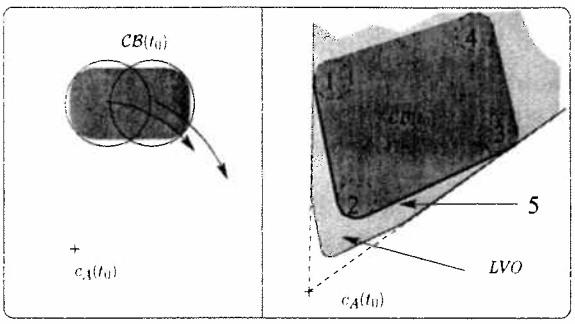

Figure 6. Approximations of a non-circular obstacle. Obstacles moving along arbitrary trajectories are approximated by conservative pavements of disks and a trajectory is assigned to each disk according to the obstacle one (left). For stationnary obstacles or obstacles with linear constant trajectories, an enclosing generalized polygon is computed such that each vertex is a disk. The disks that are in collision first (here 1, 2 and 3) and their outer tangents define the set of velocities with a time to collision equal to 1 (bold line noted as 5). The complete *LVO* is computed by scaling this line by 1/t.

a small period of time.[2] Under these assumptions, we approximate the shape of the car by a conservative pavement of disks; Each *NLVO* associated with the car and an obstacle is the union of all the single *NLVO* (expressed in the same coordinates frame) associated with a single disk and the obstacle (see Fig. 5). The accuracy of the approximation and the number of extra computation depend on the pavement chosen.

2.2.3. Non-Circular Obstacle. The shape of any obstacle $\mathcal{B}$ can be approximated by a conservative representation constructed using a pavement of disks: the trajectory of each disk can be easily derived from the trajectory of the original obstacle $\mathcal{B}$. The *NLVO* associated with $\mathcal{B}$ is defined as the union of the *NLVO* associated with each disk. However, in realistic situations, the number of disks may become too large for real-time calculations (especially for long shaped obstacles accurately approximated). Hopefully, this approximation can be drastically simplified for *LVO* (i.e. for static obstacles such as the borders of the road or for obstacles having linear constant trajectories). In such cases, $\mathcal{CB}(t_0)$ is the original obstacle $\mathcal{B}$ grown by the radius of the robot $\mathcal{A}$: a disk is centered on each edge of $\mathcal{B}$. The radius of the disks are equal to the radius of the robot. Then, the *LVO* associated to each disk is computed. The outer borders of their union define the borders of the *LVO* associated with $\mathcal{B}$ (Fig. 6).

2.2.4. Kinematics and Dynamics of the Robot. The efficiency of the *NLVO* concept lies in the simplicity of the calculations involved and the small dimension of the space $\mathcal{V}$ (or $\mathcal{V} \times \mathcal{T}$) where they are performed. The reason comes essentially from the strong hypothesis made on the trajectory followed by the robot (constant and linear on $[t_0, TH]$). When dealing with a real robot, its kinematics and dynamics rarely satisfy such constraints. A way to maintain these conditions for the computation of *NLVO* is to consider periods of time sufficiently small such that a move of the robot can be considered as constant and linear on each period. The set of all the positions that can be reached after this period of time is computed from the real kinematics and dynamics, then the set of the corresponding linear velocities V_{adm} is derived. Only the velocities in V_{adm} are considered for the computations of *NLVO* and for the choice of the future velocity.

2.2.5. Uncertainties. Uncertainty comes from perception errors and previous approximations. In order to take this uncertainty into account, security margins have to be added. Consequently, the radius of the robot and of the obstacles are grown at each time-step, according to the upper bound of uncertainty. In some cases (i.e. when the uncertainty is too high), the *NLVO* may totally cover the set of admissible velocities V_{adm}. It suggests that all the potential velocities will induce a collision on $[t_0, TH]$, however, the choice of a safe velocity is still possible, since the information on the times to collision is available (see *expansion operator* in 3.1).

3. Iterative Motion Planner

The trajectory of the robot is computed as a list of consecutive moves from its current state to its goal. A move is characterized by a constant linear velocity applied to the robot during dt seconds, the period of time between two consecutive decisions of the controller. Each move is searched in the velocity space of the robot ($\mathcal{V}$).

Our approach is based on an iterative planner in $\mathcal{V}$ and the popular A^* algorithm. A search tree is defined, such that a node n_i represents a dated state $s_A(t)$ of the robot, and a branch $b_{i,j}$ represents a safe move of dt seconds (i.e. a safe linear constant velocity $\overrightarrow{v_A}$ applied on this period) between two consecutive nodes/states:

$$\begin{cases} n_i &= \{s_A(t)\} \\ b_{i,j} &= \{\overrightarrow{v_A}\} \\ n_j &= \{s_A(t+dt) = s_A(t) + \overrightarrow{v_A} \cdot dt\} \end{cases} \quad (4)$$

The A^* algorithm considers two types of nodes: The nodes already explored, and the nodes not explored yet (called "*open*"). Exploring a node means to compute the branches issued from it using an *expansion operator* described below in Section 3.1. In our case, it consists in computing the admissible safe velocities applicable from the state of the robot associated with the explored node. Each newly created branch generates a new open node, while the last explored node is removed from the list of "*open*". Any node to be explored is chosen from this list until the goal is reached (success), the list is empty (fail) or the time available for the computation is over (timeout). In order to guaranty that an optimal trajectory among the ones explored will be found (if such a solution exists), and that the number of explored nodes will be minimal, a criteria of optimality must be chosen and estimated for each open node. We have chosen to minimise the travelling time. The method is described in Section 3.2 by the heuristic function.

3.1. Expansion Operator

The expansion of the tree consists of computing the set V_{adm} of admissible velocities according to the vehicle kinematics and dynamics. Independently, we compute the set of velocities *NLVO* that induce a collision before the given time horizon *TH* and their corresponding time to collision. *TH* depends on the vehicle velocity, the available computer ressources and for how long the obstacles trajectories prediction have been made (typical values: $1.5s \leq TH \leq 30s$).

The set of the admissible velocities that can be chosen to expand a node is almost infinite. In order to control the size of the search tree, this set is discretized, sorted and only the best five velocities are kept. The sorting is based on 2 criterias, the time to collision and the time to the goal.

3.1.1. Time to Collision. The first criteria taken into account is the safety of the robot: a risk of collision noted $Cost_{tc}(\vec{v})$ is associated with each velocity $\vec{v}$. Its value is inversely proportional to the time to collision noted $Tc(\vec{v})$. For convenience, we normalized it between 0 and 1 included, such as 0 means "*no risk*" and 1 means "*immediate collision*":

$$Cost_{tc}(\vec{v}) = \begin{cases} \dfrac{(TH - Tc(\vec{v})) \times t_0}{Tc(\vec{v}) \times (TH - t_0)} & \text{if } \vec{v} \in NLVO \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

3.1.2. Time to the Goal. The second criteria $Cost_{\text{opt}}(\vec{v})$ is based on a normalization ($Cost_{\text{opt}}(\vec{v}) \in [0, 1]$) of the travelling time to the goal, noted $T_{\text{but}}(\vec{v})$ and described later with the heuristic in Section 3.2. Its purpose is to pre-sort the safe velocities that will be chosen later by the heuristic to explore the tree:

$$Cost_{\text{opt}}(\vec{v}) = \begin{cases} 1 - \dfrac{T_{\text{but}}(\vec{v})}{t\,max_{\text{but}}} & \text{if } T_{\text{but}}(\vec{v}) \leq t\,max_{\text{but}} \\ 1 & \text{otherwise} \end{cases} \quad (6)$$

with $t\,max_{\text{but}}$ the upper bound of the time required by the robot to reach its goal.

3.1.3. Effects on the Optimality of the Trajectories. The velocities are sorted according to a cost function $Cost_{\text{global}}(\vec{v})$. It is defined as $Cost_{\text{global}}(\vec{v}) = \alpha_1 \cdot Cost_{tc}(\vec{v}) + \alpha_2 \cdot Cost_{\text{opt}}(\vec{v})$, where the α_i are real values experimentally set.

The velocities with the minimal cost are chosen to expand the node. In order to better map the free space, a velocity cannot be chosen in the neighborhood (i.e. at a fixed minimal distance in $\mathcal{V}$) of another velocity that has already been selected.

One may think that the expansion operator has an impact on the optimality of the trajectory computed from the tree, in the sense that the nodes which belong to the optimal trajectory may never be selected based on the previous criteria. This is mainly due to the assumptions on the robot linear trajectory, needed for the computation of the *NLVO*. However, this choice is very

local (an elementary move of the robot) and the eventual loss of optimal points is limited by the mapping. The heuristic search also contributes to compensate the eventual gaps between the solution computed by our approach and the one computed by a complete search. The differences observed during our tests were always negligible.

3.2. *Global Cost Function and Heuristic*

Converging quickly to a nearly optimal solution (i.e. to a trajectory that tends to minimize the travelling time in our case) implies that we are able to evaluate each open node before we choose the one to explore: A global cost function is defined as the sum of the known time needed to reach a node (number of consecutive branches from the root to the node times dt), and the estimated time needed to reach the goal from this node. This last value is our heuristic and is independant of the obstacles (which have already been taken into account by the expansion operator of the node). It is noted $T_{but}(s_A(t))$ and is computed by first estimating a simple geometrical path to the goal, according to the current robot state and its minimal turning radius (Fig. 7). A velocity profile of type "maximal acceleration-maximum speed-maximal decceleration" is computed along the geometrical path, and the corresponding travelling time $T_{but}(s_A(t))$ is deduced. This value is a good lower bound of the real travelling time and for this reason satisfies the A^* requirements, while requiring only few simple calculations.

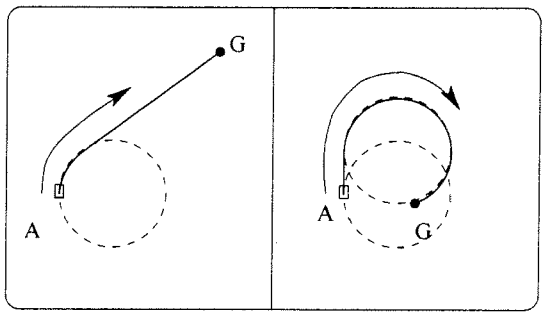

Figure 7. Geometrical Paths to the goal We consider a path composed of a segment of line and an arc of circle. Two cases are possible (The robot turns at the maximum to align with the goal then go straigth in its direction (left). When the goal is inside the minimal circle described by the robot, the robot must go straight first, then turn (right)).

3.3. *Updating the Tree*

Rebuilding the whole tree from scratch at each iteration of the controller has three consequences:

- the robot may never have time to compute a complete trajectory to the goal;
- trajectories computed at two consecutive iterations offer no guarantee to be coherent with each other;
- the same nodes may be unnecessarily explored several times at different iterations.

We propose to update the search tree instead of rebuilding it totally. Our approach is motivated by the fact that, when the predictions on the obstacles trajectories are correct, the nodes already explored (and any trajectory passing by them) do not need to be explored again at the next iteration, but should be kept to save computation time. The method is as follows: We first consider the sub-tree issued from the node that has been selected at the previous iteration (which should correspond to the current robot state). The nodes which are not part of it are deleted. In this new tree, we choose the next node to be explored from "*open*". Before exploring it, the trajectory from the root to this node is checked, starting from the root. If any collision is detected, the first node in collision and the whole sub-tree issued from it is deleted and another node is chosen in the remaining tree. Since the root of the tree changes at each iteration, we provided a function to dynamically compute the cost of each node and to guarantee that the solution is always the trajectory the most optimal among the ones explored.

By updating, the drawbacks of rebuilding a tree from scratch are avoided. Moreover, an interesting property on the robot trajectory has been observed when we do not replace immediatly the deleted nodes: In this case, the robot naturally avoids the areas where the trajectories of the obstacles had not been correctly predicted (i.e. that can be seen as area with a higher risk). In this case, the trajectories found may be less optimal, but this can be improved by associating a limited lifetime to each node, hence forcing the update of the tree.

4. Intelligent Vehicles Applications

Embedding the *NLVO* concept in an iterative planner as presented in the previous section constitutes a generic tool for motion planning with a wide field of potential applications in robotics or virtual reality

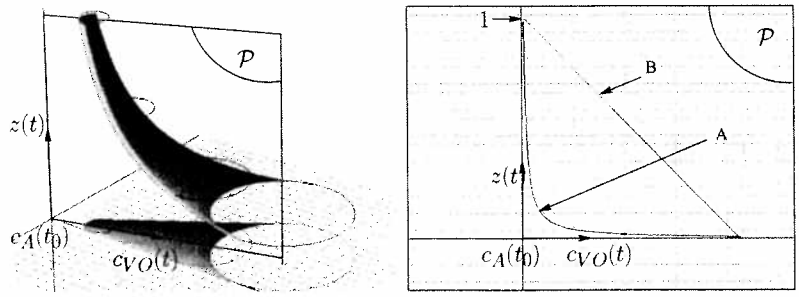

Figure 8. Linearization of the time to collision. The figure on the left is an example of a *LVO* represented in $\mathcal{V} \times \mathcal{T}$. The plan $\mathcal{P}$ is perpendicular to the plan $\mathcal{V}$ and is aligned with the axle of the *LVO*. The figure on the right represents the slope described by the middle axle of a *NLVO* (in $\mathcal{P}$). In this 2d space, the time to collision can be seen as a function of type $z(t) = c/t$ with c constant (A). B represents the slope of the same *NLVO* whom the dimension corresponding to the time-to-collision has been linearized ($z(t) = \frac{(t-t_0)\cdot TH}{(TH-t_0)\cdot t}$). Figures 4 and 9 show two examples corresponding respectively to A and B.

(see Large (2003) for several examples). We propose to demonstrate its abilities for intelligent vehicles applications.

4.1. Real-Time Issues

When navigating at high-speed among moving obstacles, the computation time is critical, the data structures and the algorithms need to be carefully designed to satisfy real-time constraints. The more important ones are presented here (see Large (2003) for complementary techniques).

4.1.1. Linearization of Time to Collision. The shape of a *LVO* in $\mathcal{V}$ is a truncated cone that can be easily constructed from only 4 points (e.g. $vo_g(t_0)$, $vo_d(t_0)$, $vo_g(TH)$ and $vo_d(TH)$). However, this is not the case in $\mathcal{V} \times \mathcal{T}$ where the straight line between $vo_g(t_0)$ and $vo_g(TH)$ (or $vo_d(t_0)$ and $vo_d(TH)$) does not match with the real border of the *LVO* (Fig. 8). In order to reduce the number of calculations, this has to be corrected using a modified scale along the time to collision axis. For all the objects represented in $\mathcal{V} \times \mathcal{T}$, we replace the coordinates associated with the time to collision t by the function $z(t)$ (Figs. 8 and 9):

$$z(t) = \frac{(t - t_0) \times TH}{(TH - t_0) \times t} \tag{7}$$

This transformation has no effect on the results: the relative sorting of the velocities, based on their times to collision, is not changed. The construction of any *LVO* becomes significantly faster (ratio of $\frac{2*dt}{TH-t_0+1}$) and the extra calculations induced by the linearization

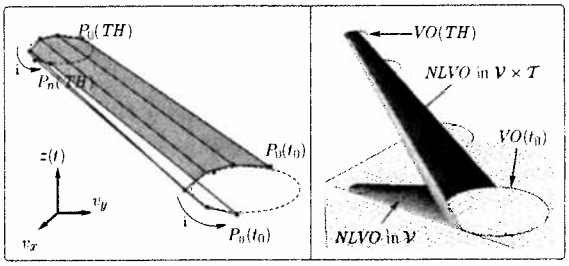

Figure 9. 3D approximation of a *NLVO* in $\mathcal{V} \times \mathcal{T}$. With the linearized time to collision, only the computation of $P_i(t)$ for $t = t_0$ and $t = TH$ is needed to represent the whole *NLVO* in $\mathcal{V} \times \mathcal{T}$. The figure shows the result for a *LVO* with $n = 49$ (right). $VO(t_0)$, $VO(TH)$ and several intermediate $VO(t)$ have been drawn (red circles) to compare with Fig. 4.

can efficiently be handled through precomputed lookup tables.

4.1.2. Drawing the NLVO. The polygonal approximation of a *NLVO* in $\mathcal{V} \times \mathcal{T}$ (Cf. Section 2.1.4) can be seen as the drawing of triangles in a 3D-space. Classical graphical libraries (e.g. OpenGL[3]) can then be used to optimize the process, especially to sort the velocities on their time to collision, and benefit from hardware acceleration when available.

First, the set of admissible velocities (V_{adm}) is computed and its complement in $\mathcal{V}$ becomes a mask in the 3D-space. The virtual camera is positionned so that the view of V_{adm} is maximized (Fig. 10). The *NLVO* are approximated by a set of triangles defined in $\mathcal{V} \times \mathcal{T}$ (Fig. 4). For convenience, the dimension of time is linearized as seen above. The triangles can now be "drawn" in the 3D-space and the time to collision of all

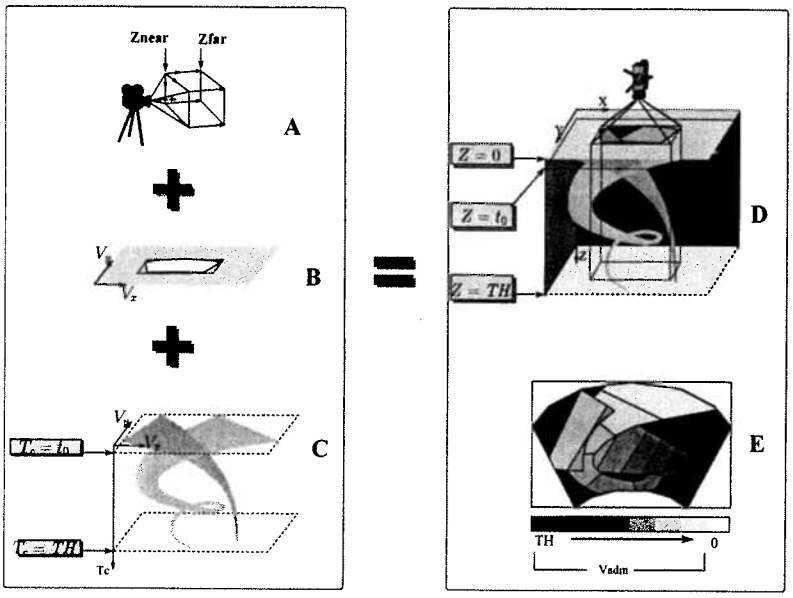

Figure 10. Computing the time to collision using a 3D library. The *NLVO* are approximated by triangles (C) drawn behind the mask of the admissible velocities (B). A virtual camera with no perception effect (A) is placed on top of the scene (D). Each pixel of the image (E) captured by the camera is associated with a depth proportional to the time to collision. A null distance denotes a non admissible velocity.

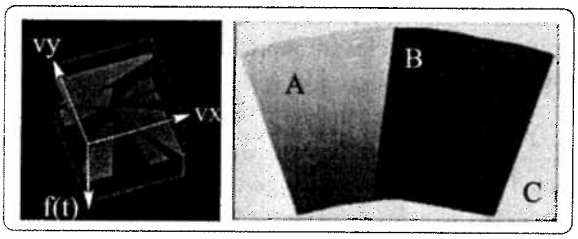

Figure 11. Examples of NLVO computed with openGL. The mask is not represented here and the camera view has been changed to show the construction of the *NLVO* in *3D* (left). The *NLVO* and the times to collision are read by positionning the camera as described in figure 10 (right). The lighter shades of red indicate a short time to collision (A), the darker red parts (B) indicate a long time to collision. The non admissible velocities (the mask) are represented in light blue (C). *Note: In order to obtain a better image, the discretization of the right captured image has been set to* 450×250. *However, the typical resolution at runtime is closer to* 32×32 *for performance reasons.*

the admissible velocities can be captured by the camera (Fig. 11). Each pixel of the "image" corresponds to an admissible linear velocity of the robot. Its Z-coordinate (usually accessible through a Z-buffer) corresponds to the time to collision associated with this velocity.

4.2. *Experimental Results*

In contrast to the majority of collision avoidance methods, we explicitly consider the movements of the obstacles. However, this implies that this information is available, otherwise the benefit of the *NLVO* is lost and navigating among fast moving obstacles may be unsafe.

To address this problem, we recently proposed a complete framework (Large et al., 2004) for a real-world problem: Navigating the parking lot of the Inria using information obtained through a number of fixed cameras covering the environment (Fig. 12). It is composed of:

- a learning-based prediction stage, which is able to produce long-term estimates of the obstacles motion in real-time,
- a motion planner as described in this paper.

Preliminary real-world experimental results have already been obtained for this first stage, but further developement needs to be done before it can be used by the planner. Meanwhile, the planner has been validated on a realistic simulated environment. We carefully designed the simulator such that the inputs of the planner (dated estimation of the future obstacles trajectories with uncertainties), the outputs of the planner (next commands of the robot), the real-time constraints (bounded computation time + delays) and the robot behaviour (kinematics and dynamics) are as close as possible to the reality.

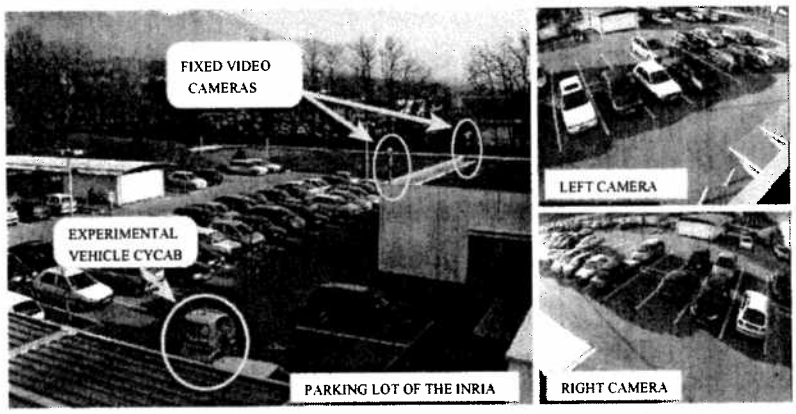

Figure 12. Experiments on the parking lot of the Inria. Left image: The parking lot of Inria (Grenoble, France), is monitored by highly placed video cameras. This infrastructure (Helin, 2003) is dedicated to the learning/prediction of the motion of the objects (vehicles and pedestrians) in this area. Right images: Examples of images captured by the cameras.

4.2.1. Simulated Environment. Several environments have been tested in simulation for intelligent vehicles, such as roads intersections, round-abouts or expressways (see Large (2003) for examples also for service robots). Our simulator is written in assembly language and C/C++ (FLTK[4] and OpenGL for graphics) and is totally dedicated to validate the *NLVO* concept. Real-life constraints such as uncertainties, limited range of perception for the robot, or undecisive behaviour of obstacles have been modelled. To do so, we defined a set of geometrical reference paths that the obstacles can follow. Each path corresponds to a particular behaviour of an obstacle. Moreover, constraints on minimum/maximum velocities, temporary breaks or complete stops can be defined along the path. A simulation session starts by randomly setting the parameters associated with the obstacles. These parameters are the instant when the obstacle will appear, the reference path it will follow (i.e. its behaviour), and with which accuracy it will do it. For each obstacle, a trajectory is then computed, so that it will not collide with the obstacles that have already been created and which are still on the road. When a complete environment is generated for a period of time sufficiently long, the simulation is launched. It consists in creating a robot on demand with an initial state and a final state. The robot must find its way from one state to the other without colliding into any static obstacle (borders of the road) or any moving obstacle (other vehicles). When several robots are present at the same time on the road, each one considers the other as an obstacle and needs to estimate its trajectory. In the example presented later, we extracted the geometrical path from the current computed trajectories of each robot. A new velocity profile is then applied to this path, assuming that each robot will keep constant its current linear velocity. This introduces errors that oblige the robots to update their predictions. When the future trajectories are not available, the future moves of the obstacles are approximated from their last previous moves.

4.2.2. Example of an Expressway Junction. The presented example shows a dangerous junction on an expressway. Vehicles can enter, exit, or continue on the same lane. Figure 13 illustrates a car-like robot (red) adapting its speed to enter safely on the expressway. Another car-like robot (blue) does the same to continue on the main lane. The other vehicles are created as mentionned above (predefined behaviour). This example illustrates a case of passive cooperation between the two cars and illustrates how each car can react in real-time to changes in the environment: The blue car follows a smooth trajectory, that can be easily predicted by the red one. Hence, the red adapts its speed to the blue one which does not need to modify its own speed. On the other hand, the blue car may not necessary consider the red one as a potential danger in the beginning since the estimated future trajectory at this time is not the real one. Hence, the blue car "concentrates" on its goal and goes straight at a maximum velocity. The acceleration of the blue car in order to reach its maximal velocity obliges the red car to increase its own velocity. This has an effect on the blue car which needs to deccelerate a bit to let the red car pass. After the red

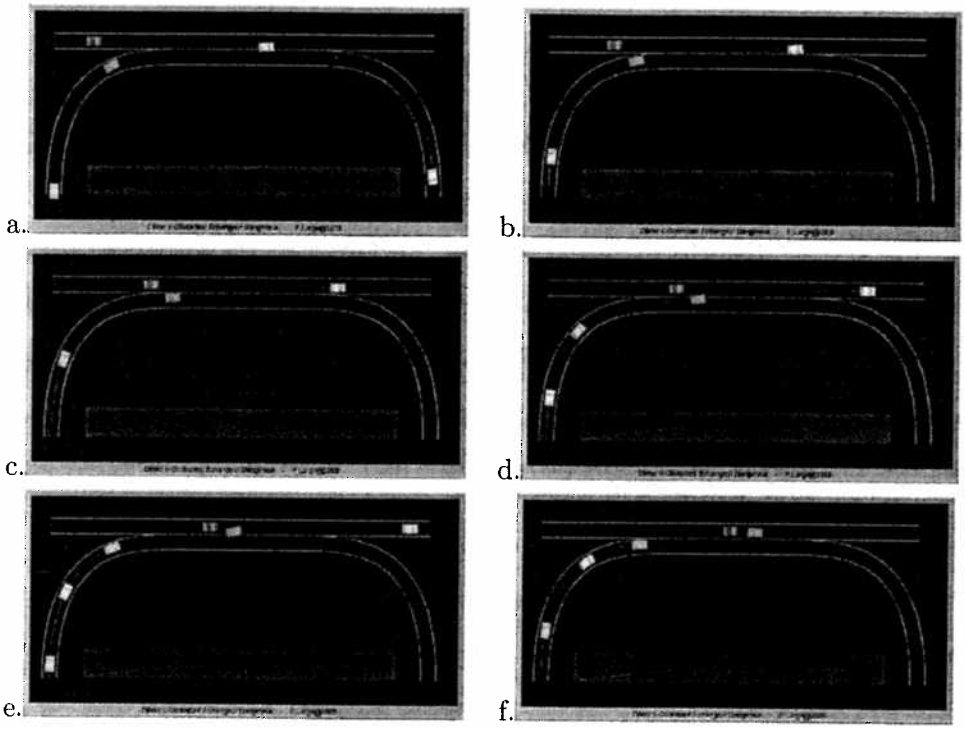

Figure 13. Navigation Example. See comments in the text. Images are read from left to right, top to bottom.

car has merged on the left lane, both cars accelerate in order to reach their maximal velocity.

5. Conclusion and Perspectives

In this paper, we first reviewed the principles of the *NLVO* concept and its adaptation to handle realistic situations. This concept is then embedded into an iterative planner to compute safe trajectories to a goal among moving obstacles. The approach demonstrated for a challenging realistic traffic scenario. The main advantages of this method are the following: the robot reacts quickly to any change in its environment; the computed trajectories avoid dangerous areas of the environment where the predictions are not correct; the future trajectories of the obstacles are taken into account to better anticipate the collisions; simple kinematics and dynamics of the controlled vehicle are taken into account; a risk of collision is computed in real-time for any potential move of the robot.

The next step is to implement the planner on real robots or vehicles. Firstly, we need a method to predict the obstacles trajectories. We are now working on the integration of such a method and our planner in a complete framework (Large et al., 2004): A delimited area (road intersection, carpark) is monitored by fixed video cameras (Helin, 2003) to learn typical motion patterns of the obstacles. The future trajectory of each obstacle is deduced from the prior observations and serves as input for the motion planning stage.

Future work will aim at expressing the *NLVO* in a reference frame that would "follow" a nominal trajectory of the robot. This would allow the approach to better take into account the kinematics/dynamics of the robot, and is particulary interesting for intelligent vehicles.

Acknowledgments

This work has been partially supported by the LaRA (La Route Automatisée) and the IMARA (Informatique, Mathématiques et Automatique pour la Route Automatisée) programs of INRIA on intelligent road transport systems, both directed by M. Michel Parent. The authors would like to thank Dr. Sepanta Sekhavat for his contributions to this work, M. Dizan Vasquez-Govea for his work on motion prediction, and Dr. Thierry Fraichard and Mrs. Priscilla Large for proof-reading the paper.

Notes

1. Induced errors can be compensated by growing the robot shape if needed. Other approximation methods can be found in Large (2003) when the robot motion cannot be approximized by a sequence of pure translations.
2. The period of the robot controller in our case.
3. 'Open Graphical Library' (OpenGL): http://www.opengl.org
4. 'Fast Light ToolKit' (FLTK): http://www.fltk.org

References

Burke. 1985. *Applied Differential Geometry*. Cambridge University Press.

Brock and Khatib. 1999. High speed navigation using the global dynamic window approach. In *Proc. Int. Conf. on the Robotics and Automation.*

Brock and Khatib. 2000. Real-time replanning in high-dimensional configuration spaces using sets of homotopic paths. In *Proc. Int. Conf. on the Robotics and Automation*, San Francisco, USA.

Erdmann, M. and Lozano-Perez, T. 1987. On multiple moving objects. *Algorithmica*, (2):477-521, also appeared in *Proceedings of IEEE International Conference on Robotics and Automation*, San Francisco, CA, pp. 1419–1424.

Fiorini, P. and Shiller, Z. 1993. Motion planning in dynamic environments using the relative velocity paradigm. In *IEEE International Conference on Robotics and Automation*, Atlanta GA.

Fiorini, P. and Shiller, Z. 1998. Motion planning in dynamic environments using the relative velocity obstacles. *IEEE International Journal of Robotics Research*, 17(7):760–772.

Fox, D., Burgard, W., and Thrun, S. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33.

Fraichard, Th. 1999. Planning in a dynamic workspace: A state-time space approach. *In Advance Robotics*, 13(1):75–94.

Frazzoli, E., Dahleh, M.A., and Feron, E. 2002. Real-time motion planning for agile aautonomous vehicles. *AIAA Journal of Guidance, Control and Dynamics*, 25(1):116–129.

Fujimura, K. and Samet, H. 1989. A hierarchical strategy for path planning among moving obstacles. *IEEE Trans. Robotics and Automation*, 5(1):61–69.

Helin, F. 2003. 'Développement de la Plate-forme Expérimentale Parkview pour la Reconstruction de l'Environnement Dynamique. *mémoire de fin d'études*, INRIA, Grenoble, France.

Hsu, D., Kindel, R., Latombe, J.-C., and Rock, S. 2000. Randomized kinodynamic motion planning with moving obstacles. In *Algorithmic and Computational Robotics: New Directions: The Fourth International Workshop on the Algorithmic Foundations of Robotics*. B.R. Donald et al. (Eds.), pp. 247–264.

Ko, N.Y. and Simmons, R. 1998. The lane-curvature method for local obstacle avoidance. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).*

Large, F. 2003. Navigation autonome D'Un robot mobile En environnement dynamique Et incertain. University of Savoie, France.

Large, F., Vasquez-Govea, D-A., Fraichard, Th., and Laugier, Ch. 2004. High-speed navigation among unknown moving obstacles. in *Proc. of the IEEE Intelligent Vehicles Symposium*, Parma, Italy.

Minguez, J. and Montano, L. 2000. Nearness diagram navigation. A new real-time collision avoidance approach. *IEEE/RSJ Int. Conf on Intellingent Robots and Systems*, Takamatsu, JP.

Minguez, J., Montano, L., Siméon, N., and Alami, R. 2002. Global nearness diagram navigation. In *Proc. of the IEEE Int. Conf on Robotics and Automation*, Washington, DC.

Shiller, Z., Large, F., and Sekhavat, S. 2001. Motion planning in dynamic environments: Obstacles moving along arbitrary trajectories. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Seoul, Korea, pp. 3716–3721.

Stachniss, C. and Burgard, W. 2002. An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems.*

Ulrich and Borenstein. 1998. VFH+: Reliable obstacle avoidance for fast mobile robots. In *IEEE Int. Conf. on Robotics and Automation*. Leuven, Belgium.

Since 1998, **Frédéric Large** is a member of the French National Consortium on road automation, namely LaRA ("La Route Automatisée"), in the eMotion (formely known as Sharp) research project-team at INRIA (French National Institute for Research on Computer Science and Control). He holds an engineer degree in artificial intelligence (1998) and a master and his Ph.D. in computer science (2003) from the University of Savoie in the Rhône-Alpes (France). Dr. Large works on modelling and reasonning for mobile service-robots and intelligent vehicles. In particular, he specialises on autonomous navigation in environments shared by moving obstacles whose future behaviour is uncertain.

Christian Laugier is Research Director at INRIA (French National Institute for Research on Computer Science and Control), and leader of the robotics project-team at INRIA Rhône-Alpes since 1984. He received the Ph.D. and "State Doctor" degrees in Computer Science from Grenoble University (France) in 1976, and 1987 respectively. His current research interests mainly lies in the areas of Motion Autonomy, Intelligent Vehicles, Decisional Processes, and Virtual

Reality. In 1997, he was awarded the Nakamura Prize for his contribution to the advancement of the technology on Intelligent Robots and Systems. Pr. Christian Laugier is a member of several scientific national and international committees (several French scientific committees, Adcom of IROS, Adcom of the EURON European Network, etc.), and he is regularly involved in the organizing committees of the major international conferences in Robotics (e.g. IEEE ICRA and IEEE/RSJ IROS). In addition to his research and teaching activities, he participated in the start-up of four industrial companies in the fields of Robotics, Computer Vision, and Computer Graphics.

Zvi Shiller is a Professor, founder and head of the Department of Mechanical Engineering-Mechatronics at the College of Judea and Samaria in Ariel, Israel. He received the B.Sc. ('76) from Tel Aviv University, and the M.Sc. ('84) and Sc.D. ('87) from the Massachusetts Institute of Technology, all in Mechanical Engineering. Before joining the College of Judea and Samaria in 2001, he served fourteen years on the faculty of the Department of Mechanical and Aerospace Engineering at UCLA. Professor Shiller's current research interests include trajectory planning, multi-robot coordination, navigation of autonomous off-road vehicles and active safety of intelligent road vehicles.