Avoiding Cars and Pedestrians Using Velocity Obstacles and Motion Prediction

Frédéric Large, Dizan Vasquez, Thierry Fraichard & Christian Laugier
Inria Rhône-Alpes
http://www.inrialpes.fr/sharp

Abstract— **Vehicle navigation in dynamic environments is an important challenge, especially when the motion of the objects populating the environment is unknown. Traditional motion planning approaches are too slow to be applied in real-time to this domain, hence, new techniques are needed. Recently, iterative planning has emerged as a promising approach. Nevertheless, existing iterative methods don't provide a way to estimating the future behaviour of moving obstacles and to use the resulting estimates in trajectory computation. This paper presents an iterative planning approach that addresses these two issues. It consists of two complementary methods: 1) A motion prediction method which learns typical behaviours of objects in a given environment. 2) An iterative motion planning technique based on the concept of Velocity Obstacles.**

I. Introduction

To some extent, autonomous vehicle navigation in stationary environments is no longer a problem. The challenge now is autonomous navigation in environments containing moving obstacles and especially moving obstacles whose future behaviour is unknown. In the presence of moving obstacles, reasoning about their future behaviour is required. When this future behaviour is unknown, one has to resort to predictions and autonomous navigation faces then a double constraint: constraint on the response time available to compute a motion (which is a function of the dynamicity of the environment), and constraint on the temporal validity of the motion planned (which is a function of the validity duration of the predictions). In other words, one needs to be able to plan motions fast but one does not need to plan motion very far in the future.

Autonomous navigation approaches are classically split between motion planning approaches (wherein a complete motion to a goal is computed once, e.g.[1], [2]), and reactive ones (wherein only the next move is computed, e.g.[3], [4]). Planning approaches are too slow whereas reactive ones have too little look-ahead. Accordingly, none of them are satisfactory when confronted to unknown moving obstacles.

So-called iterative planning approaches have appeared lately [5], [6], [7]. They account for the two constraints mentioned above and *iteratively* compute a *partial motion* at a given frequency. Instead of computing the next move only, several steps are computed depending on the time available. Different possibilities are explored and a partial trajectory is incrementally built. They can be interrupted at any time so as to keep the vehicle reactive, while the trajectory returned is the best among the ones explored in the allocated time.

This work has been partially supported by a Conacyt scolarship. We also want to thank the support of the CNRS Robea ParkNav and the Lafmi NavDyn Projects.

Such approaches are the most promising. Nevertheless, they require two important conditions that are not satisfied in current methods yet: the future behaviour of the moving obstacles must be estimated, and this estimation must be taken into account in the partial trajectory computation.

This paper presents an iterative planning approach that addresses these two issues. The case of an autonomous vehicle evolving in a confined environment observed by video cameras is considered. The two issues, i.e. obstacles motion prediction and vehicle motion planning are dealt with by two complementary methods:

Obstacles motion prediction The environment is monitored by video cameras in order to learn the typical motions of the moving obstacles. Once the learning stage is completed, the future motion of any given obstacle can be predicted.

Vehicle motion planning The concept of *Velocity Obstacle* [8] is used to estimate efficiently the safety of a vehicle's motion in the 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 out. The tree is updated to reflect the environment changes every time a trajectory is computed.

Obstacles motion prediction and vehicle motion planning are respectively detailed in §II and §III. Preliminary experimental results are presented in §IV.

II. Obstacles Motion Prediction

The motion prediction technique we propose operates in two stages: a learning stage and an estimation stage. This structure is common to a number of relatively recent proposals that also try to learn typical motion patterns, e.g. [9], [10].

The training data used in the learning stage consists in a set of N obstacles trajectories. In our case, the trajectories were obtained by means of video cameras monitoring the environment considered [11]. A trajectory $d_i, i = 1 \dots N$, is a time sequence of moving obstacles configurations: $d_i = \{q_1, ..., q_{T_i}\}$ where T_i is the total number of captured configurations for the i^{th} trajectory. In this paper, it is assumed that the q_j represent the obstacles position (x, y), and that they are evenly sampled in time (so that the moving obstacles velocities are intrinsically represented too).

Training data is clustered and each resulting cluster is considered to represent a typical motion pattern. For each cluster obtained,we compute a representative trajectory: the mean value of all the trajectories in the cluster, and its standard deviation. Since we have used the velocity information to perform the clustering, the mean value is, effectively, a trajectory and not just a geometrical path.

In the estimation stage, we model the likelihood that a partially observed trajectory belongs to a given cluster as a Gaussian probability function. The parameters of that function are the mean value and standard deviation that we have found in the learning stage. We compute this likelihood for all the clusters. The estimated motion is given by the mean value of the trajectory having a maximum of likelihood. An alternative could be to use all the motion patterns having a likelihood greater than a given threshold.

A. Learning Algorithm

In order to discover the typical motion patterns, we analyse training data. We expect that trajectories which are very similar correspond to objects engaged on the same motion pattern. Thus, we will try to find groups of similar trajectories. This leads quite naturally to the use of a clustering algorithm.

A.1 Clustering Trajectories

The selection of a particular clustering technique is somewhat difficult because the best one to be used depends on the particular problem considered [12]. We have chosen a formulation which does not confines itself to the utilisation of a single algorithm, so that different clustering techniques can be tested in order to find the one that produces the best results.

Many clustering algorithms [12], [13] are able to work using a dissimilarity matrix, which is an $n \times n$ matrix containing all the pairwise dissimilarities[1] between n objects. Thus, finding a way to measure dissimilarities between trajectories allows us to use any of those algorithms.

A trajectory d_i can be viewed as a piecewise defined function of time $d_i(t)$ consistent of straight segments. We define the dissimilarity, or distance between two trajectories d_i and d_j as:

$$\delta(d_i, d_j) = \left(\frac{1}{\max(T_i, T_j)} \int_{t=0}^{\max(T_i,T_j)} \left(d_i(t) - d_j(t)\right)^2 dt \right)^{1/2} \quad (1)$$

Where T_i and T_j are the total motion duration of d_i and d_j respectively, and $d(t) = d(T)$ for $t > T$. This function is the average Euclidean distance between two functions, we have chosen the average because we want our measure to be independent of the length of the trajectories being compared.

[1]Dissimilarities result from comparing two objects: their value is high if the compared objects are very different, and is zero if they are identical. They are always nonnegative [13]

Using (1), we can construct a dissimilarity matrix and use it as the input for a clustering algorithm to obtain a clustering consisting on a set of clusters C_k represented as lists of trajectories.

A.2 Calculating Cluster Mean-Value and Standard Deviation

One drawback of pairwise clustering is that, as it operates directly over the dissimilarity table, it does not calculate a representation of the cluster. So, if we want to use the cluster's representation as an estimate, we have to calculate this representation.

We have chosen to represent each cluster using what we call its mean-value. Let C_k be a cluster having N_k trajectory functions $d_i(t) \mid 1 \leq i \leq N_k, d_i(t) \in C_k$ then, we define the mean value of C_k as the following function:

$$\mu_k(t) = \frac{1}{N_k} \sum_{i=1}^{N_k} d_i(t) \quad (2)$$

Calculating the standard deviation for the cluster C_k using the mean value is straightforward using the following expression:

$$\sigma_k = \left(\frac{1}{N_k} \sum_{i=1}^{N_k} \delta(d_i, \mu_k)^2 \right)^{1/2} \quad (3)$$

Once we have calculated both the mean value and standard deviation for each cluster, we can use those parameters to estimate motion by applying a criterion of Maximum Likelihood as explained next.

B. Estimation Algorithm

The output of the learning algorithm consists of a list of mean values and standard deviations corresponding to the different typical behaviours detected.

In order to estimate trajectories, we calculate the likelihood of a partially observed trajectory d_p under each one of the clusters. To do that, we model classes as Gaussian sources with the mean value and standard deviation that were calculated during learning.

B.1 Partial Distance

As we are dealing with partial trajectories, we need to modify (1) to account for this. The modification consists in measuring the distances respect to the duration of the partial trajectory:

$$\delta_p(d_p, d_i) = \left(\frac{1}{T_p} \int_{t=0}^{T_p} \left(d_p(t) - d_i(t)\right)^2 dt \right)^{1/2} \quad (4)$$

Where δ_p, d_p and T_p are the partial distance, partially observed trajectory, and duration of the partial trajectory, respectively.

B.2 Calculating Likelihood

With the partial distance (4), we can directly estimate the likelihood that d_p belongs to a cluster C_k.

$$P(d_p \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k}} e^{-\frac{1}{2\sigma^2}\delta_p(d_p,\mu_k)^2} \quad (5)$$

Once we have calculated the likelihood, we can choose, for example, to estimate the trajectory using the mean value of the cluster with maximal likelihood, or to present the different possibilities having likelihood greater than a given threshold.

III. Robot motion planning

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 robot. In this context, a trajectory will be represented by the linear velocities applied to the robot at each iteration of the controller, and will be searched in the velocity space of the robot ($\mathcal{V}$). This space allows a 2-D geometric (i.e. fast) representation of the admissible instantaneous linear velocities of the robot according to its kinematics, its dynamics, and the obstacles (through the NLVO formalism introduced later).

Our approach is based on an iterative planner in this space and the popular A^* algorithm. A search tree is defined, such as 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.

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

The A^* algorithm considers two types of nodes: The nodes already explored, and the nodes not explored yet (called the "*open*"). Exploring a node means to compute the branches issued from it using an *expansion operator* described later. 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 computations is out (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 criterion of optimality must be chosen and estimated for each open node. We chose the travelling time as the criterion to minimize, and defined an heuristic function described later to estimate it.

A. Expansion operator

The expansion operator consists first in computing the set V_{adm} of admissible velocities according to the robot kinematics and dynamics.

Independently, we compute the set of velocities *NLVO* that induce a collision before a given time horizon noted TH[2].

The set of safe admissible velocities $\vec{v}$ is such as $\vec{v} \in V_{adm}$ et $\vec{v} \ni NLVO$. The problem is to compute *NLVO* efficiently.

Concept of Non-Linear V-Obstacle We defined the concept of *Non-Linear V-Obstacle* (*NLVO*) in [14] as the set of all the linear velocities of the robot, that are constant on a given time interval $[t_0,\ TH]$ and that induce a collision with an obstacle before TH. We call $\mathcal{A}$ the robot, $\mathcal{B}_i$ an obstacle and $\mathcal{V}$ the velocity space of $\mathcal{A}$:

$$NLVO = \bigcup \left\{ \vec{v} \in \mathcal{V} \;\middle|\; \exists\ t \in [t_0,\ TH],\ \mathcal{A}(t) \cap \mathcal{B}_i(t) \neq \emptyset \right\}$$

Computing NLVO From a geometric approach, a *NLVO* can be seen in $\mathcal{V}$ as a set of ribbons each corresponding to an obstacle. In [14], we proposed an analytical expression of the borders of these ribbons. In [15], we extended $\mathcal{V}$ by a dimension of time corresponding to the time to collision ($\leq TH$) associated with each linear velocity. The ribbons (*NLVO*) are then defined in this 3-D space, noted $\mathcal{V} \times \mathcal{T}$ (figure 1).

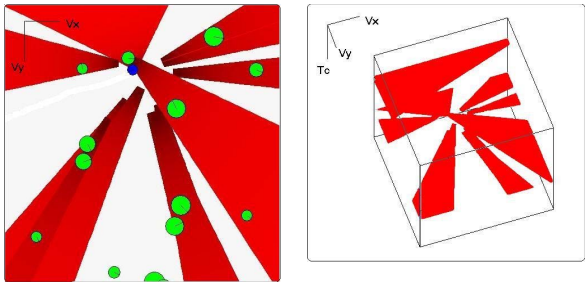

Fig. 1. *NLVO examples* In the $\mathcal{V}$ space (green disks represent the obstacles, the blue one is the robot, and the red shapes (one by obstacle) constitute the *NLVO*). The expression of the *NLVO* in $\mathcal{V} \times \mathcal{T}$ give extra information on the time to collision associated with each velocity.

Their construction allows a fast estimation of the velocities that will induce a collision and the corresponding time to collision (Please refer to [15] for details).

Selecting velocities for exploration The set of velocities that can be chosen to expand a node is virtually infinite and depends on TH. In order to control the size of the search tree, this set is discretized, sorted and only the 5 best velocities are kept.

The first criterion 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 $Tc(\vec{v})$ and is normalized for convenience:

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

The second criterion $Cost_{opt}(\vec{v})$ is based on a normalization of the travelling time to the goal, noted $T_{but}(\vec{v})$ and

[2] TH depends on the robot velocity, the available computer ressources and for how long the obstacle trajectories prediction have been made. We chose $1.5s \leq TH \leq 30s$

described later with the heuristic. Its purpose is to pre-sort the safe velocities and to keep only the more susceptible to be chosen later by the heuristic used to explore the tree.

$$Cost_{opt}(\vec{v}) = \begin{cases} 1 - \frac{T_{but}(\vec{v})}{tmax_{but}} & \text{if } T_{but}(\vec{v}) \leq tmax_{but} \\ 1 & \text{otherwise} \end{cases}$$

The velocities are then sorted according to a global cost $Cost_{global}(\vec{v})$ defined by $Cost_{global}(\vec{v}) = \alpha_1 \cdot Cost_{tc}(\vec{v}) + \alpha_2 \cdot Cost_{opt}(\vec{v})$, where the α_i are real values experimentally set.
The velocities having minimal distances are chosen to expand the node, but in order to better map the free space, a minimal distance is defined such that two velocities are always sufficiently far from each other.

B. Heuristics

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 one to be explored: An heuristic function is defined as the summation 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 noted $T_{but}(s_A(t))$ and is computed by first estimating a simple geometric path to the goal, according to the current robot state and its minimal turning radius (figure 2).

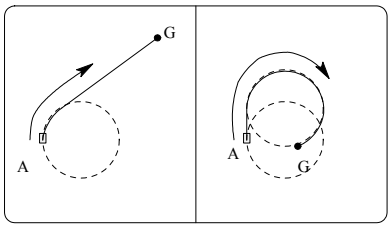

Fig. 2. *Geometric Paths to the goal* We consider a path composed of a segment and an arc of circle. Two cases are possible: The robot turns at the maximum to align with the goal then go straigth in its direction (left figure). When the goal is inside the minimal circle described by the robot, the robot must go straight first, then turn (right figure).

A velocity profile of type "maximum acceleration-maximum speed-maximum decceleration" is then computed and the corresponding travelling time $T_{but}(s_A(t))$ is deduced. This value appears to be a good lower bound of the real travelling time and for this reason satisfies the A^* requirements, while requiring only a few simple calculations.

C. Updating the tree

Rebuilding the whole tree from scratch at each iteration of the controller implies 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) don't need to be explored again at the next iterations but should be kept instead to save computation time. The idea is to keep the sub-tree issued from the node selected at the previous iteration (which should correspond to the current robot state). The next node to be explored is then chosen from its list of open, previously computed, as if the tree has just been built. The difference leads in the fact that before actually exploring the selected node, the trajectory linking it to the root is checked. If any collision is detected, starting from the root, then the node and the whole sub-tree issued from it is deleted and another node is chosen.

All the drawbacks of a complete rebuild listed earlier are canceled by this method. Moreover, an interesting property on the robot trajectory has been observed: it naturally avoids the areas where the trajectories of the obstacles had not been correctly predicted (hence that present a higher risk). On the other end, the trajectories may be less optimal, but this can be improved by associating a limited lifetime to each node, and forcing by this way the update of the tree.

IV. Experimental Results

We have implemented and tested our motion estimation technique using two clustering algorithms: Agglomerative Complete-Link Hierarchical Clustering (CL) [16] and Deterministic Annealing Pairwise Clustering (DA) [17]. To validate our motion estimation technique, we have performed a series of tests using data coming from two environments: a trajectory simulator, and a pedestrian tracking system placed in the Inria lobby (fig. 3a)). An estimation example can be seen in figs. 3c-d.

As for our planning approach, we present in fig. 4 a navigation example using a simulator. The figure illustrates

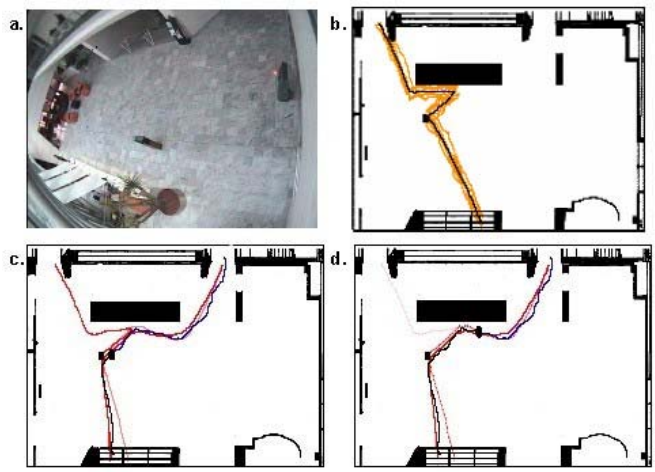

Fig. 3. a) The Inria entry hall. b) Trajectory Cluster. c, d) Subsequent Predictions

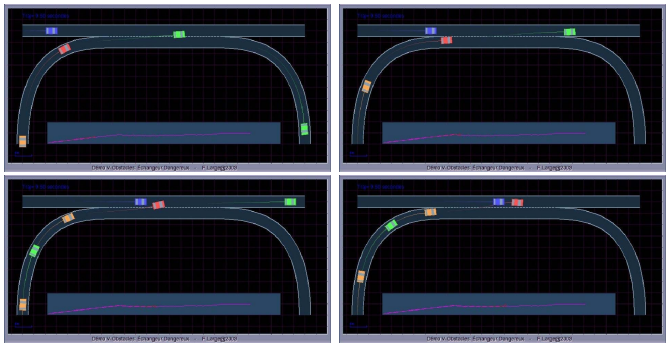

Fig. 4. Navigation Example

a car-like robot adapting its speed to enter safely on an exchanger.

We are now working in the integration of the framework and in its application to a real world problem: Navigating the parking lot of the Inria using information obtained through a number of fixed cameras covering the environment.

References

[1] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," A.I. Memo 883, MIT AI Lab., Boston, MA (USA), May 1986.

[2] K. Fujimura and H. Samet, "Time-minimal paths among moving obstacles," in *In IEEE International Conference on Robotics and Automation*, (Scottsdale, AZ (USA)), pp. 1110–1115, May 1989.

[3] Ulrich and Borenstein, "Reliable obstacle avoidance for fast mobile robots," in *IEEE Int. Conf. on Robotics and Automation*, (Victoria, Canada), 1998.

[4] 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 (IROS)*, (Victoria, Canada), 1998.

[5] T. Fraichard, "Trajectory planning in a dynamic workspace: a 'state-time' approach," *Advanced Robotics*, vol. 13, no. 1, pp. 75–94, 1999.

[6] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," in *Workshop on the Algorithmic Foundations of Robotics, 2000*, 2000.

[7] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA Journal of Guidance, Control and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[8] F. Large, S. Sekhavat, Z. Shiller, and C. Laugier, "Towards real-time global motion planning in a dynamic environment using the NLVO concept," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Lausanne, Switzerland), 2002.

[9] E. Kruse, R. Gusche, and F. M. Wahl, "Acquisition of statistical motion patterns in dynamic environments and their application to mobile robot motion planning," (Grenoble, France), pp. 713–717, 1997.

[10] M. Bennewitz, W. Burgard, and S. Thrun, "Learning motion patterns of persons for mobile service robots," in *Proceedings of the 2002 IEEE Int. Conf. On Robotics and Automation*, (Washington, USA), pp. 3601–3606, 2002.

[11] F. Helin, "Développement de la plate-forme expérimentale parkview pour la reconstruction de l'environnement dynamique," mémoire de fin d'études, Conservatoire Nat. des Arts et Métiers, Grenoble (FR), July 2003.

[12] A. Jain, M. Murty, and P. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, pp. 265–322, September 1999.

[13] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series In Probability And Mathematical Statistics, John Wiley and Sons, Inc., 1989.

[14] Shiller, Large, and Sekhavat, "Motion planning in dynamic environments: Obstacles moving along arbirtrary trajectories," in *Proceedings of the IEEE Int. Conf. On Robotics and Automation*, (Seoul, Korea), pp. 3716–3721, 2001.

[15] F. Large, *Navigation Autonome D'un Robot Mobile En Environment Dynamique et Incertain*. PhD thesis, Université de Savoie, 2003.

[16] B. King, "Step-wise clustering procedures," *Journal of the American Statistical Association*, vol. 69, pp. 86–101, 1967.

[17] T. Hofmann and J. M. Buhmann, "Pairwise data clustering by deterministic annealing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 1–14, 1997.