Moving Obstacles' Motion Prediction for Autonomous Navigation

Dizan Alejandro Vasquez Govea, Frédéric Large, Thierry Fraichard, Christian Laugier

2004 8th International Conference on
Control, Automation, Robotics and Vision
Kunming, China, 6-9th December 2004

Moving Obstacles' Motion Prediction for Autonomous Navigation

Dizan Vasquez, Frédéric Large, Thierry Fraichard & Christian Laugier

Inria Rhône-Alpes
http://emotion.inrialpes.fr

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 do not provide a way to estimate the future behavior of moving obstacles and 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 behaviors of objects in a given environment. 2) an iterative motion planning technique based on the concept of Velocity Obstacles.

1 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 behavior is unknown. In the presence of moving obstacles, reasoning about their future behavior is required. When this future behavior 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 time during which the predictions are sound). 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.

Such approaches are the most promising. Nevertheless, they require two important conditions that are not satisfied in current methods yet: the future behavior 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 an environment observed by video cameras is considered. The two issues, *ie* 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 moving 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 §2 and §3. Preliminary experimental results are presented in §4.

2 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 tra-

jectories 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 a moving object is tracked, and the likelihood that its trajectory observed so far belongs to a given cluster is calculated. The estimated motion is given by the mean value of the cluster having a maximum likelihood. An alternative could be to use all the motion patterns having a likelihood greater than a given threshold.

2.1 Learning Algorithm

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

2.1.1 Clustering Trajectories

The selection of a particular clustering technique is somewhat difficult because the best one to be used depends on the problem at hand [12]. We have chosen a formulation which does not confines itself to the utilization 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 between the n objects. 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]. 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 function which returns the object configuration as a function of time, $d_i(t) = q_t$ for $t \in [0, T_i]$, and the dissimilarity, or distance between two trajectories d_i and d_j is defined 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 is assumed that $T_i < T_j$ and $d_i(t) = d_i(T_i)$ for $t > T_i$. 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.

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

2.1.2 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 $d_i(t)$. The mean value of C_k is defined as:

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

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.

2.2 Estimation Algorithm

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

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

2.2.1 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:

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

Where d_o and T_o are the trajectory observed so far and its duration, respectively.

2.2.2 Calculating Likelihood

With the partial distance (4), we can directly estimate the likelihood that d_o belongs to a cluster C_k using a gaussian probability distribution.

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

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.

3 Iterative Motion Planner

The future 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 (*ie* a safe linear constant velocity $\overrightarrow{v_A}$ applied on this period) between two consecutive nodes/states:

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

The A^* algorithm considers two types of nodes: The nodes already explored, and the nodes not explored yet (called "*open*"). Exploring a node means to compute the branches issued from it using an *expansion operator* described below in 3.2. 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 guarantee 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. The criterion to minimize the traveling time is defined by the heuristic function presented in 3.3. When a node is explored, the concept of *Non-Linear $\mathcal{V}$-Obstacle* described in 3.1 is used to reduce the computation time.

3.1 *Concept of* Non-Linear $\mathcal{V}$-Obstacle

We defined the concept of *Non-Linear $\mathcal{V}$-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 $\mathcal{A}$:

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

From a geometrical standpoint, 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], the time dimension (corresponding to the time to collision) was added to $\mathcal{V}$. The ribbons (*NLVO*) are then defined in this 3-D space, noted $\mathcal{V} \times \mathcal{T}$ (fig. 1). Classical graphical libraries (*eg* openGL [16]) can then be used to optimize the computation and benefit from hardware acceleration when available.

The construction of the *NLVO* in $\mathcal{V} \times \mathcal{T}$ allows a fast estimation of the velocities that will induce a collision and the corresponding time to collision (please refer to [15] for details).

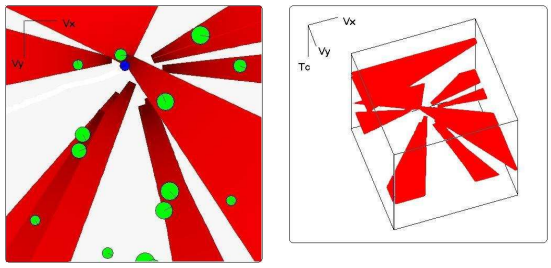


Fig. 1. *NLVO examples* In $\mathcal{V}$ (left) the green disks represent the obstacles, the blue one is the robot, and the red shapes (one by obstacle) constitute the *NLVO*. The expression of the *NLVO* in $\mathcal{V} \times \mathcal{T}$ (right) gives an extra information on the time to collision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

3.2 *Expansion Operator*

The expansion of the tree consists in 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 (See 3.3 for the method). TH depends on the vehicle velocity, the available computer resources and for how long the obstacle 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 theoretically infinite. In order to control the size of the search tree, this set is discretized, sorted and only the five best velocities are kept. Sorting is based on two criteria: time to collision and time to the goal.

3.2.1 Time to Collision

The first criterion taken into account is the safety of the robot: For each velocity $\vec{v}$, we compute a risk of collision noted $Cost_{tc}(\vec{v})$. Since a collision in a short time is potentially more dangerous than a collision in a further time, a simple expression of $Cost_{tc}(\vec{v})$ could be $Cost_{tc}(\vec{v}) = \frac{1}{Tc(\vec{v})}$ with $Tc(\vec{v}) \in [t_0; TH]$. For convenience, we prefer to map $Cost_{tc}(\vec{v})$ into $[0; 1]$, where 0 would denote an immediate collision (at t_0). This leads to a slightly more complicated expression of $Cost_{tc}(\vec{v})$:

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

3.2.2 Time to the goal

The second criterion $Cost_{opt}(\vec{v})$ is based on a normalization of the traveling time to the goal, noted $T_{goal}(\vec{v})$ and described later with the heuristic in 3.3. Its purpose is to pre-sort the safe velocities and only keep the more susceptible to be chosen later by the heuristic used to explore the tree:

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

The velocities are then sorted according to a global cost function noted as $Cost_{global}(\vec{v})$ and defined as $Cost_{global}(\vec{v}) = \alpha_1 \cdot Cost_{tc}(\vec{v}) + \alpha_2 \cdot Cost_{opt}(\vec{v})$, where the α_i are real values experimentally set.

The velocities 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 (*ie* at a fixed minimal euclidean distance in $\mathcal{V}$) of another velocity that has already been selected.

3.3 Heuristic

Converging quickly to a nearly optimal solution (*ie* to a trajectory that tends to minimize the traveling time in our case) implies that we are able to evaluate each open node before we choose one to be explored: A heuristic 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 noted $T_{goal}(s_A(t))$ 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. 2). A velocity profile of type "maximum acceleration-maximum speed-maximum deceleration" is computed along the geometrical path, and the corresponding traveling time $T_{goal}(s_A(t))$ is determined. This value is a good lower bound of the real traveling time and for this reason satisfies the A^* requirements, while requiring only few simple calculations.

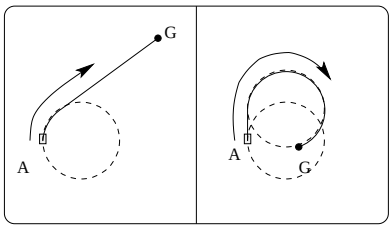


Fig. 2. *Geometrical Paths to the goal* we consider a path composed of a segment of line and an arc of circle. Two cases are possible: The robot turns to align with the goal then go straight in its direction (left). When the goal is inside the minimal circle described by the robot, the robot must go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

3.4 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 iterations 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. Valid nodes are explored as described in 3.2.

By updating, the drawback of rebuilding a tree from scratch is avoided. 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 (*ie* with a higher risk). The computed trajectories 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 Experimental Results

In order to validate our techniques we have performed a number of tests in different environments. In this section we describe and comment our experiments for both motion prediction and planning. Finally we provide an overview of our current work on a real system installed on the parking lot of our institute.

4.1 Motion Prediction

We have used data coming from two environments: a trajectory simulator and a pedestrian tracking system placed in the Inria entry hall (fig. 3). The tracking system installation is underway. Hence, our main testbed is the simulated environment, which recreates pedestrian motion in the Inria entry hall (fig. 3).

For the simulated environment, we have generated two sets of data: training data and test data. We have used the training data to learn the motion patterns, and then, we have used the test dataset to evaluate the obtained results using two clustering algorithms: Complete-Link agglomerative clustering (CL)[17] and Deterministic Annealing (DA)[18](fig. 3).

In order to test the performance of our approach we have also implemented another technique based

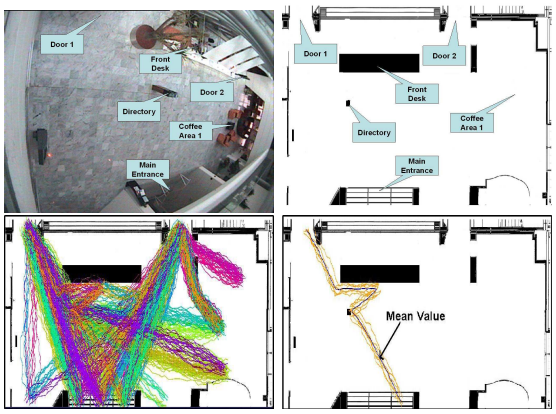
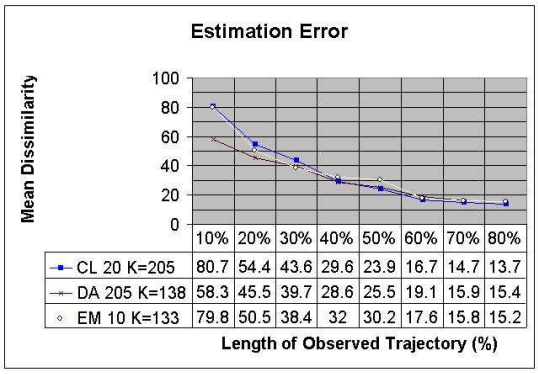


Fig. 3. Top: The INRIA entry hall and the simulated environment. Bottom: Raw trajectories and an example cluster.

on the Expectation-Maximization (EM) algorithm [10]. To get a performance metric, we measure the difference between estimated and real trajectories. For each trajectory in the test dataset we take a fraction of its total length. Using this fraction, we search for a match in the set of clusters obtained in the learning stage. The selected cluster will be that having the highest likelihood. We calculate the estimation error as the distance between the mean value of the selected cluster and the complete real trajectory. The error is measured for trajectory lengths between 10% and 80% of the complete trajectory. This procedure is repeated for each of the clustering methods.

The results of our tests can be seen in fig. 4. We can observe that, for all the techniques, we get better predictions as we use longer observed trajectories. Another observation is that, when we know more than 30% of the total trajectory, performance for all the techniques gets very similar and can be considered quite accurate for the kind of motion being analyzed. Finally, we verify that, in our experiments, our technique performed slightly better than the Expectation-Maximization approach.

Our unoptimized implementation of the technique is able to produce estimates with a frequency of $60 - 100Hz$, which we consider adequate for real-time systems involving vehicles and pedestrians.



	10%	20%	30%	40%	50%	60%	70%	80%
CL 20 K=205	80.7	54.4	43.6	29.6	23.9	16.7	14.7	13.7
DA 205 K=138	58.3	45.5	39.7	28.6	25.5	19.1	15.9	15.4
EM 10 K=133	79.8	50.5	38.4	32	30.2	17.6	15.8	15.2

Fig. 4. Estimation errors for different techniques.

As a result of our experiments, we have shown that our technique is able to learn motion patterns from observations and to produce sound, long-term motion estimates in real time.

4.2 Iterative Motion Planning

Experiments on real vehicles require a complex infrastructure not available yet and preliminary experiments on motion prediction and planning have been carried out in simulation.

Our motion planner has been tested on various simulated scenarios, such as roads intersections, round-abouts or expressways. The example depicted in fig. 6 shows a dangerous junction on an expressway, where vehicles can enter, exit, or continue on the same lane. A car-like robot (red) is 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 follow predefined known trajectories, however, the trajectories used in the $NLVO$ calculations are estimated from previous states only. This example illustrates a case of passive cooperation between the two robots and illustrates how each robot can react in real-time to changes in the environment: The blue robot 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 see the red one as a potential danger at the beginning since its estimated future trajectory at this time is not the real one. Hence the blue car can "concentrate" on its goal and go straight at maximal velocity. Later on, 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 decelerate a bit to let the red car pass. After merging, both cars accelerate in order to reach their maximal velocity.

4.3 Parking Lot Experiments

We are now working on 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 (fig. 7).

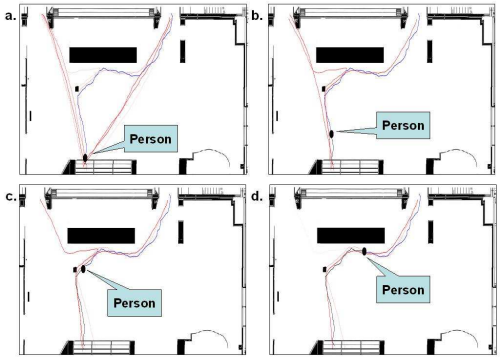


Fig. 5. Motion hypothesis at different moments.

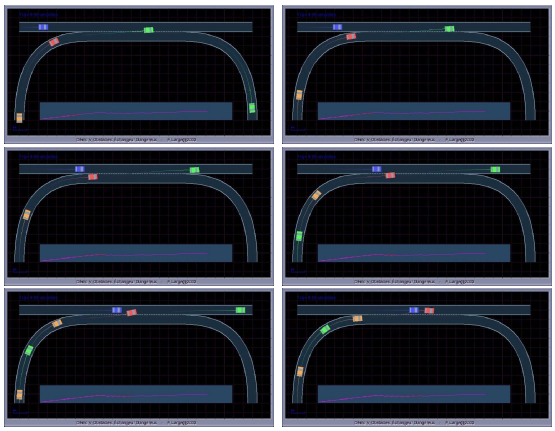


Fig. 6. *Navigation Example* See comments in the text. Images are read from left to right, top to bottom.

5 Conclusions

In this paper we have proposed two techniques which can be applied in order to solve the navigation problem in a dynamic environment:

- A learning-based estimation technique which is able to produce long-term estimates of the motion of heterogeneous objects in real time.
- An iterative motion planning technique which is based on the concept of Non-Linear Velocity Obstacles which adapts its planning scope with respect to the available time.

Future work includes the possibility to include information on the environment's state to produce more accurate predictions; and further experimentation with the tracking system installed on Inria's parking lot.

Fig. 7. Inria's parking lot. Overall view (left) and seen from a camera (right).

Acknowledgements

This work has been partially supported by a Conacyt scholarship. We also want to thank the support of the CNRS Robea ParkNav, the Lafmi NavDyn and the European Cybercars projects. The authors would like to thank Prof. Zvi Shiller for his contribution to this work.

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 Int. Conf. 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*, (Leuven, Belgium), 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*, (Hanover, NH (USA)), 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," in *Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, (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 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] "Opengl graphical library," *http://www.opengl.org*.

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

[18] 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.