High-Speed Autonomous Navigation with Motion Prediction for Unknown Moving Obstacles
Dizan Alejandro Vasquez Govea, Frédéric Large, Thierry Fraichard, Christian Laugier

To cite this version:

HAL Id: inria-00182062
https://inria.hal.science/inria-00182062
Submitted on 24 Oct 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 behaviour 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 behaviours of objects in a given environment. 2) an iterative motion planning technique based on the concept of Velocity Obstacles.

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.

Keywords — Dynamic Motion Planning, Trajectory Estimation, Iterative Planning
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 behaviour 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 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.

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 Obstacles [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 $d_i, i = 1 \ldots N$, is a time sequence of moving obstacles configurations: $d_i = \{q_{1i}, \ldots, q_{Ti}\}$ where $T_i$ is the total number of captured configurations for the $i^{th}$ trajectory. In this paper, it is assumed that the $q_i$ represent the obstacles position $(x, y)$, and that they are evenly sampled in time (so that the moving obstacles velocities are intrinsically repre-
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 a moving object is tracked, and the likelihood that its trajectory observed so far belongs to a given cluster is modelled 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 function of time, and the dissimilarity, or distance between two trajectories \(d_i\) and \(d_j\) is defined as:

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

(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.

\(^1\)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 \(d_i(t)\). The mean value of \(C_k\) is defined as:

\[
\mu_k(t) = \frac{1}{N_k} \sum_{i=1}^{N_k} d_i(t)
\]

(2)

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

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

(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 trajectory observed so far \(d_o\) under each one of the clusters. To do that, we model behaviours 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:

\[
\delta_o(d_o, d_i) = \left( \frac{1}{T_o} \int_{t=0}^{T_o} (d_o(t) - d_i(t))^2 \, dt \right)^{1/2}
\]

(4)

Where \(d_o\), \(\delta_o\) and \(T_o\) are the trajectory observed so far, its distance and its duration, respectively.
B.2 Calculating Likelihood

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

\[
P(d_p \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k}} e^{-\frac{1}{2\sigma_k^2}(d_p - \mu_k)^2}
\]

(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. 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 (\( V \)).

Our approach is based on an iterative planner in \( V \) 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 \( \overline{v_A} \) applied in this period) between two consecutive nodes/states:

\[
\begin{align*}
  & n_i = \{ s_A(t) \} \\
  & b_{i,j} = \{ \overline{v_A} \} \\
  & n_j = \{ s_A(t+dt) = s_A(t) + \overline{v_A} \cdot dt \}
\end{align*}
\]

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 III-B. 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 criteria of optimality must be chosen and estimated for each open node. The criteria to minimise the travelling time is defined by the heuristic function presented in III-C. When a node is explored, the concept of Non-Linear \( \mathcal{V} \)-Obstacle described in III-A is used to reduce the computation time.

A. 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 \( \mathcal{A} \):

\[
NLVO = \{ v \in \mathcal{V} \mid \exists \ t \in [t_0, TH],\ \mathcal{A}(t) \cap \mathcal{B}_i(t) \neq \emptyset \}
\]

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 (e.g. 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).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig1.png}
\caption{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.}
\end{figure}

B. 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 III-C 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.

B.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 \( Tc(\vec{v}) \) and is normalized for convenience (0="no risk" to 1="immediate collision"):
The acceleration \( \text{Cost}_{\text{te}}(\vec{v}) \) is computed along the geometrical path, and for this reason satisfies the \( A^* \) requirements, while requiring only few simple calculations.

### B.2 Time to the goal

The second criteria \( \text{Cost}_{\text{opt}}(\vec{v}) \) is based on a normalization of the travelling time to the goal, noted \( T_{\text{bat}}(\vec{v}) \) and described later with the heuristic in III-C. Its purpose is to pre-sort the safe velocities and only keep the more susceptible to be chosen later by the heuristic to explore the tree:

\[
\text{Cost}_{\text{opt}}(\vec{v}) = \begin{cases} 
1 - \frac{T_{\text{bat}}(\vec{v})}{T_{\text{max}}(\vec{v})} & \text{if } T_{\text{bat}}(\vec{v}) \leq T_{\text{max}}(\vec{v}) \\
0 & \text{otherwise}
\end{cases}
\]

The velocities are then sorted according to a global cost function noted as \( \text{Cost}_{\text{global}}(\vec{v}) \) and defined as

\[
\text{Cost}_{\text{global}}(\vec{v}) = \alpha_1 \cdot \text{Cost}_{\text{te}}(\vec{v}) + \alpha_2 \cdot \text{Cost}_{\text{opt}}(\vec{v}),
\]

where the \( \alpha_i \) are real values experimentally set.

### C. 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 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 \( T_{\text{bat}}(s_A(t)) \) 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 "maximal acceleration-maximum speed-maximal deceleration" is computed along the geometrical path, and the corresponding travelling time \( T_{\text{bat}}(s_A(t)) \) is deduced.

### IV. 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.

#### A. 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 and Deterministic Annealing (fig. 4).

In order to test the performance of our approach 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.

We have found that both techniques produce estimates having a mean error of around 30 cm for input trajectories of 40% of the total length, which can be considered quite accurate for the kind of motion being analyzed.

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

We have also implemented another technique [10] in order to benchmark our approach. In our experiments, our technique performed slightly better than the other approach. 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.

B. 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, roundabouts or expressways. The example depicted in fig.7 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 necessarily 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.

C. 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 (fig. 8) of the Inria using information obtained through a number of fixed cameras (fig. 9) covering the environment.

V. 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.

REFERENCES


