HAL
open science

Safe Vehicle Navigation in Dynamic Urban Scenarios

Kristijan Macek, Dizan Alejandro Vasquez Govea, Thierry Fraichard, Roland Y. Siegwart

To cite this version:

Kristijan Macek, Dizan Alejandro Vasquez Govea, Thierry Fraichard, Roland Y. Siegwart. Safe Vehicle Navigation in Dynamic Urban Scenarios. IEEE Conference on Intelligent Transportation Systems, Oct 2008, Beijing, China. inria-00326267

HAL Id: inria-00326267

https://inria.hal.science/inria-00326267v1

Submitted on 2 Oct 2008

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.

Safe Vehicle Navigation in Dynamic Urban Scenarios

Kristijan Maček*, Dizan Vasquez*, Thierry Fraichard†, Roland Siegwart*

*Swiss Federal Institute of Technology Zürich (CH), †Inria Grenoble Rhône-Alpes (FR)

I. Introduction

A. Background and Motivations

Autonomous mobile robots/vehicles navigation has a long history by now. Remember Shakey's pioneering efforts in the late sixties[1]. Today, the situation has dramatically changed as illustrated rather brilliantly by the 2007 DARPA Urban Challenge[1]. The challenge called for autonomous car-like vehicles to drive 96 kilometers through an urban environment amidst other vehicles (11 self-driving and 50 human-driven). Six vehicles finished the race thus proving that autonomous urban driving could become a reality. Note however that, despite their strengths, the Urban Challenge vehicles have not yet met the challenge of fully autonomous urban driving (how about handling traffic lights or pedestrians for instance?).

Another point worth mentioning is that at least one collision took place between two competitors. It was nothing serious but it raises the important issue of *motion safety*, *ie* the ability for an autonomous robotic system to avoid collision with the objects of its environment. With robotic systems designed to operate in the real world among human beings in many cases, motion safety becomes critical. The size and the dynamics of the Urban Challenge vehicles make them potentially dangerous for themselves and their environment (especially when driving at high-speed). Therefore, before letting such autonomous systems move among people, it is vital to assert their operational motion safety.

In the last forty years, the number and variety of autonomous navigation schemes that have been proposed is huge (*cf* [2]). In general, these navigation schemes aims at fulfilling two key purposes: reaching a goal while avoiding collision with the objects of the environment. When it comes to collision avoidance, once again, many collision avoidance schemes have been proposed. Their aim of course is to ensure the robotic systems' safety. However the analysis carried out in [3] of the most prominent navigation schemes (*ie* the ones currently used by robotics systems operating in real environments, *eg* [4], [5], [6], [7]) shows that, especially in dynamic environments, *motion safety is not guaranteed* (in the sense that it is easy to find situations where collisions will eventually occur). To some extent, this is due to the fact that safety is a concept that is taken for granted. In other words, the meaning of safety is never formally stated and, above all, the operational conditions of such collision avoidance schemes are seldom (if never) spelled out.

B. Contributions and Paper Outline

In the past few years, the Autonomous Systems Lab. of the Swiss Federal Institute of Technology have been active with self-driving cars through the SmartTer initiative[2]. It has developed a navigation architecture that has evolved over the years. The primary purpose of this paper is to present the latest developments of the deliberative part of this navigation architecture. These developments are geared towards autonomous driving in dynamic urban environments with a particular focus on the motion safety issue.

The deliberative part of the architecture features two key modules working together in a hierarchical fashion: the *Route Planner* (high-level) and the *Partial Motion Planner* (low-level).

The purpose of the Route Planner is to provide the Partial Motion Planner with a valid route towards a given goal. A route in this case comprises a geometric path augmented with additional information based on the structure of the environment considered. Such a route should comply with the standard regulations for vehicles driving in a urban setting. This means that factors such as speed limits and stop signs should be taken into account.

It is up to the Partial Motion Planner to take care of all the gory details of the actual driving. It relies upon the route and a local model of the vehicle's environment (with up to date information about the fixed and the moving objects) in order to determine the next motion command to apply to the vehicle. As the name suggests, a Partial Motion Planning scheme is used [8], [9] in order to (a) take into account the *decision time constraint* imposed by dynamic environments and (b) improve convergence towards the desired goal. Motion safety is dealt with by the Partial Motion Planner. Two safety levels respectively called *Passive* and *Passive Friendly* are explored and their operational conditions are explicitly stated.

The paper is organized as follows: §II briefly overviews the complete navigation architecture. The Route Planner and the Partial Motion Planner are respectively presented in §III and §IV. §V details the particular diffusion technique used by the Partial Motion Planner whereas §VI focuses on the safety issues. The experimental platform is overviewed in §VII and preliminary navigation results are finally presented in §VIII.

[1] `http://www.darpa.mil/grandchallenge`.

[2] `http://www.smart-team.ch`

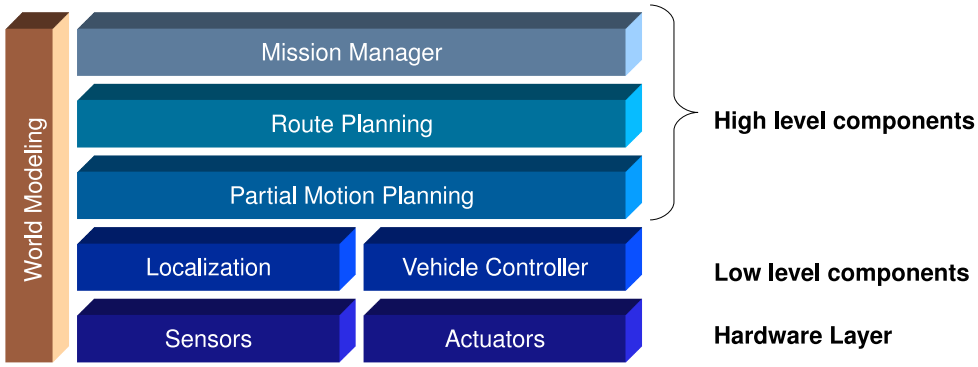

Fig. 1. Overview of the proposed navigation architecture.

II. Navigation Architecture

Fig. 1 presents an overview of the navigation architecture of the SmartTer platform. It is structured in layers, where, in most cases, higher level components interact with all the lower level layers in the hierarchy. The only exception is the world model, which is directly accessed by all the levels of hierarchy. Given that this paper focuses on route planning and partial motion planning we will just briefly discuss the high-level components and their interaction here.

1) Mission Manager. It is responsible of translating high-level tasks (*eg* pick-up a person at an address) into initial and a goal configurations that the vehicle should reach in order to accomplish those tasks.
2) Route planning. It is concerned with finding a global route for the vehicle between the initial and goal configurations given by the mission manager, using information about the static characteristics of the environment (e.g. lane geometry, speed limits). We assume that knowledge about these characteristics is available *a priori* so that it is possible to perform at least part of the computations off-line.
3) Partial Motion Planning. It is responsible of executing routes computed by the route planning module including collision avoidance and –for unstructured environments– motion planning. It integrates knowledge about the dynamic elements of the environment,and interacts tightly with the world model.
4) World model. The world model gathers all the available information about the environment, including the vehicle's localization, the environment structure and the current and predicted states of the other objects that are present in the environment. As mentioned above, it is different from other modules because it interacts with all the levels of the hierarchy.

The next two sections respectively describe Route Planning and Partial Motion Planning.

III. Route Planning

The goal of route planning is to exploit prior knowledge about the environment in order to provide local navigation with a feasible and valid path between two points of the environment. In this context, valid means that the path should be collision free and comply with traffic regulations for vehicles driving in a urban setting, where factors such as speed limits and stop signs should be taken into account by the planner.

More specifically, prior knowledge comes in the form of a slightly augmented[3] Route Network Definition File (RNDF)[10]. This information may be seen as a directed graph (see fig 2) where we distinguish between two types of nodes, *waypoints*, having a special meaning (*eg* an intersection or a parking entrance) and standard nodes which are used as a piecewise linear approximation to describe higher order curves. Additionally, unstructured, open areas of the environment are described as polygons having entry and exit points that are connected to other nodes of the graph.

From this graph, route planning is straightforward by applying the A* graph search algorithm using the length of the graph's edges as a cost function. Once a path has been found, it is smoothed out in order to guarantee its feasibility. The output of the module consists of a list of configurations $\mathcal{Q}_g = \{q_g^1, \cdots, q_g^{N_g}\}$ that the vehicle should reach. Each configuration can be described as a position, orientation and curvature:

$$q_g^i = \{x_g^i, y_g^i, \theta_g^i, \kappa_g^i\} \tag{1}$$

[3]RNDF's have been augmented by including speed limits.

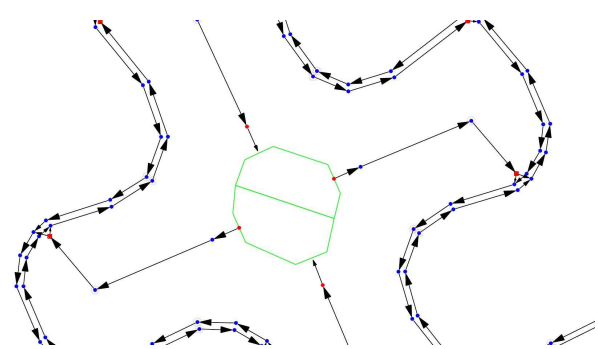

Fig. 2. Detail of final DARPA's RNDF file showing waypoints (red), standard nodes (blue) and unstructured areas (green polygons).

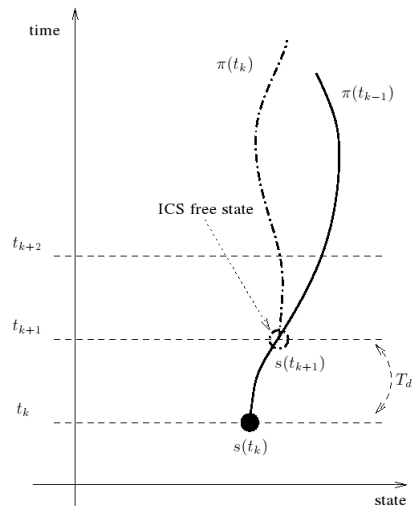

Fig. 3. Partial motion planning iterative scheme

Associated with every configuration, there is a constraint vector describing the bounds within which the vehicle's motion is considered as acceptable with respect to the task at hand and the traffic rules (*eg* speed limits), as well as a Boolean flag wp^i, whose value is true when the configuration corresponds to a waypoint and false otherwise:

$$c^i = \{r^i_{\max}, \Delta\theta^i_{\max}, \Delta\kappa^i_{\max}, v^i_{\min}, v^i_{des}, v^i_{\max}, wp^i\} \quad (2)$$

where $r^i_{\max}$ stands for the maximum distance between the object's actual position and the desired one; $\Delta\theta^i_{\max}$ and $\Delta\kappa^i_{\max}$ are the maximum error tolerated for the heading angle and the curvature, respectively; and v^i_{min}, v^i_{des} and $v^i_{\max}$ are the minimum, desired and maximum velocities associated to the given configuration.

IV. Partial Motion Planning

The Partial Motion Planner (PMP) is the core navigational module of the architecture. Its primary purpose is to determine the motion command u that is sent to the vehicle controller at every time cycle. The motion command u must meet the following requirements:

- *Feasibility:* it must take into account the dynamic constraints of the vehicle;
- *Goal Convergence:* it must eventually drive the vehicle towards the desired goal.
- *Safety:* it must ensure the safety of the vehicle, *ie* its ability to avoid collisions.

PMP operates iteratively with a time period T_d which is determined by the environment (in an environment featuring moving objects, you have a limited time only to decide upon your future course of action otherwise you run the risk of being hit by a moving object). T_d constitutes the *decision time constraint*.

To determine u, PMP (as the name suggests) applies the Partial Motion Planning principle [8]: it tries to make the best possible use of the decision time T_d available by computing a partial motion towards the goal. To that end, a diffusion technique is used to explore the state$\times$time space of the vehicle and determine a partial motion π that is used during the next time cycle to drive the vehicle towards its goal. Fig. 3 illustrates how PMP operates. Let t_k denote the current time instant and the beginning of the kth PMP cycle. The previous PMP cycle has computed the partial motion $\pi(t_{k-1})$ that starts at time t_k. The kth PMP cycle then has to compute $\pi(t_k)$ that will start at time $t_{k+1} = t_k + T_d$. The process is repeated until the goal is reached.

At every cycle, PMP takes as input an updated model of the environment that comprises:

- the route computed by the Route Planner, *ie* the list $\mathcal{Q}_g = \{q^1_g, \cdots, q^{N_g}_g\}$ and the corresponding constraints $c^i, i = 1 \cdots N_g$ (*cf* §III),
- a list $\mathcal{O}_s$ of static objects: it is assumed that the static part of the environment is known from the road network structure and is described by a list of forbidden regions represented by closed polygons;
- a list $\mathcal{O}_d$ of dynamic objects: one fundamental task of the World Modelling module is to provide PMP with an updated model of the environment of the vehicle at every time cycle. PMP must know what are the moving objects present in the environment and, most important, what their future behaviour will be. To that end, the World Modelling module features a Prediction module whose purpose is to estimate the future behaviour of the moving objects. It is assumed that the prediction is valid over a given *prediction horizon* of duration T_p. The moving objects are described by a list of forbidden regions whose position varies over time. Their shape is modelled by rectangular bounding boxes which is suitable for vehicles and pedestrians alike (more general shapes could be used). The notation $\mathcal{O}_d(t)$ is used to indicated the fact that their position varies over time.

Each partial motion π computed respects the dynamic constraints of the vehicle considered thus meeting the Feasibility requirement. By nature, PMP aims at maximizing the lookahead of the navigation process (the exploration of the future is carried out as far as possible given the decision time T_d available). In our opinion, this is one way to meet the Goal Convergence requirement. Finally, each partial motion π computed will be safe in a predefined way (for instance, it will be guaranteed that the vehicle always have the possibility to brake down and stop before a collision occurs). This is the answer to the Safety requirement.

The next two sections respectively describe the diffusion technique used and how motion safety is handled.

V. Diffusion Technique

The partial motion, *ie* the trajectory, that is to be computed for the vehicle can be described as a single parametric curve, *eg* polynomial or spline curve, or a concatenation of several geometrical primitives such as arcs or clothoids.

The kinematic vehicle model is described by the Ackermann model:

$$\dot{x} = \cos\theta\, v_l\,, \qquad \dot{y} = \sin\theta v_l\,, \qquad \dot{\theta} = \frac{v_l}{L}\tan\phi\,, \tag{3}$$

with $\{x, y, \theta\}$ being the robot pose and $\{v_l, \phi\}$ the longitudinal velocity and steering angle. Therefore the full vehicle state at the current navigation cycle t_k can be described as:

$$s(t_k) = \{x(t_k), y(t_k), \theta(t_k), v_l(t_k), \phi(t_k)\} \tag{4}$$

The dynamic update of the system can be described in the discrete general form:

$$\dot{s}(t_k) = f(s(t_k), u_d(t_{k-1})) \tag{5}$$

where the dynamic level control input vector u_d is the longitudinal acceleration $\dot{v}_l(t_{k-1})$ and the steering rate $\dot{\theta}(t_{k-1})$. The dynamic update function f encapsulates the physical dynamic model of the vehicle, including inertia and physical forces acting on the vehicle itself.

If a low-level control is implemented separately (cascade control) which handles directly the actuators of the vehicle, *ie* gas pedal (longitudinal acceleration $\dot{v}_l$) and steering wheel torque (steering rate $\dot{\theta}$), the system function f represents the closed-loop response of the vehicle with low-level control. Therefore, the actual commands issued from the Partial Motion Planning level are the kinematic control reference values:

$$u = \{v_{l,ref}, \phi_{ref}\} \tag{6}$$

The control vector u is derived directly from the planned trajectory π at each navigation cycle. Moreover, if the latency of the low-level control is very small in comparison to the navigation cycle T_d, the closed-loop vehicle response to a kinematic control reference value u can be described with circular arcs (*ie* if the steering angle ϕ_{ref} is kept fixed during time T_d. Using arc geometric primitives largely reduces the computational costs that would be incurred if the full numerical integration through the system function f would be performed for possible input u.

In order to explore different possible trajectories, the geometric arc primitives are concatenated in a randomized sampled diffusion scheme, where the trajectory structure grows in a RRT ("Rapidly-Exploring Random Tree") manner [11]. The motion exploration phase starts with the current vehicle state $s(t_k)$) and the control input (trajectory) from the previous cycle $u(t_{k-1})$, the goal configuration(s) $\mathcal{Q}_g$, the set of dynamic obstacles $\mathcal{O}_d$(t) and static obstacles $\mathcal{O}_s$. The new exploration states are inserted into a search tree $\mathcal{T}$ as can be seen in Table I. The available decision time for exploration is of duration T_d, after which a valid trajectory with associated control inputs $u(t_k) = \{v_{l,ref}(t_k), \phi_{ref}(t_k)\}$ is returned or failure is signalled.

Firstly, the newly formed tree $\mathcal{T}$is initialized by its root node τ_{root}, where the information contained in each node is of the form:

$$\tau = \{s, u, w_\tau, t_\tau, d_\tau\} \tag{7}$$

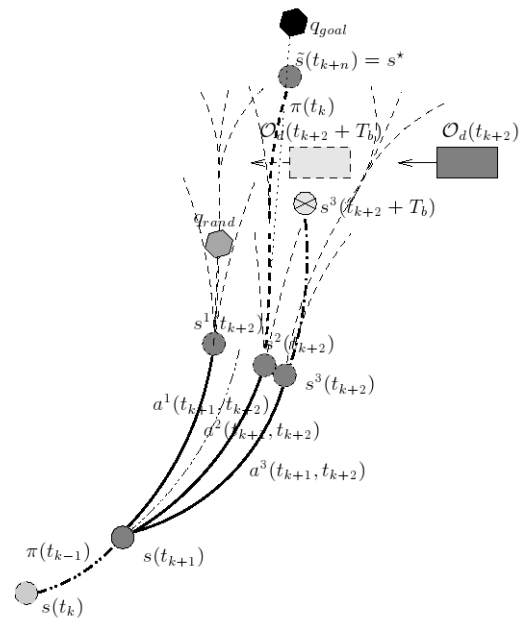

Fig. 4. Node expansion

The s represents the state of the vehicle, u the reference input that induced the state s, w_τ the overall cost to reach the node τ, t_τ the cumulative time with respect to the root of the tree $\mathcal{T}$ and d_τ the node depth within the tree. Therefore, τ_{root} which contains the predicted state $s(t_{k+1})$ (PREDICT_STATE) for the given control input $u(t_{k-1})$ from the previous navigation cycle and zeroed cost, cumulative time and tree depth. The state prediction for the root node is necessary due to the fact that the currently computed trajectory $\pi(k)$ will be available only at the end of the current navigation cycle (see Fig.3). The $\mathcal{L}_1$ contains all the nodes that are appended to the current tree depth $d_\mathcal{T}$. The overall computation time t_s can only be within the T_d span (Table I.L5).

There are three tree expansion methods implemented in the current PMP scheme:

1) *exploration*: expansion towards a random configuration in the environment - q_{rand} (Table I.L9-L13);
2) *goal search*: expansion towards a goal configuration - q_{goal} (Table I.L14-L18);
3) *exhaustive search*: extension from a given tree node (state) using a randomized control input - u_{rand} (Table I.L19-L23).

These cases are depicted further in Fig. 4. In case 1, the predicted state $s(t_{k+1})$ is expanded towards a random configuration q_{rand} in the workspace ($s^1(t_{k+2})$), whereas in case 2 the tree $\mathcal{T}$ is expanded towards the goal configuration q_{goal} ($s^2(t_{k+2})$). In both cases the function EXTEND_STATE_SPACE first finds the node τ_{near} in the tree

```
PMP_SEARCH(s(t_k), π(t_{k-1}), Q_g, O_d(t), O_s, T_d)
1  t_s=0.0, d_T=0;
2  s(t_{k+1}) ← PREDICT_STATE(s(t_k), π(t_{k-1}));
3  τ_root.init(s(t_{k+1}), u(t_{k-1}), 0.0, 0.0, d_T);
4  T.init(τ_root), L_1.init(τ_root), L_2=∅, τ_⋆=∅;
5  while t_s ≤ T_d do
6      for n=1 to |L_1|
7          τ_ext ← L_1.pop();
8          p ← RAND();
9              if (p < T.p_r)
10                 q_g^rand ← RAND_STATE_SPACE(O_s);
11                 τ_new ← EXTEND_STATE_SPACE(q_g^rand,...);
12                     if not (τ_new = ∅) then
13                     INSERT_WITH_COST(τ_new, Q_g, T, L_2, d_T, τ_⋆);
14             if (T.p_r ≤ p < T.p_r + T.p_g)
15                 q_g^rand ← RAND_SELECT(Q_g);
16                 τ_new ← EXTEND_STATE_SPACE(q_g^rand,...);
17                     if not (τ_new = ∅) then
18                     INSERT_WITH_COST(τ_new, Q_g, T, L_2, d_T, τ_⋆);
19             else
20                 u_rand ← RAND_COM_SPACE(u(t_{k-1}));
21                 τ_new ← EXPAND_COM_SPACE(τ_ext, u_rand,...);
22                     if not (τ_new = ∅) then
23                     INSERT_WITH_COST(τ_new, Q_g, T, L_2, d_T, τ_⋆);
24     end
25 d_T=d_T+1;
26 swap(L_1, L_2);
27 UpdateTime(t_s);
28 end
29 if not τ_⋆ = ∅ then
30     return PATH(T, τ_⋆);
31 else return failure;
```

```
EXTEND_COM_SPACE(τ, u, O_d, O_s)
32 if (τ.d_τ== 0) then
33     τ_new ← EXTEND_WITH_SAFETY_CHECK(τ, u,O_d, O_s);
34 else
35     τ_new ← EXTEND_WITH_COLLISION_CHECK(τ, u, O_d, O_s);
36 if not (τ_new==∅) then
37     τ_new.d_τ=τ.d_τ+1;
38     T← T∪ τ_new;
39     return τ_new;
40 else return ∅;
```

```
EXTEND_STATE_SPACE(q_g, T, O_d, O_s)
41 τ_near ← NEAREST_NEIGHBOR(q_g, T);
42 u_near ← NEAREST_COMMAND(τ_near, q_g);
43 return  EXTEND_COM_SPACE(τ_near, O_d, O_s);
```

```
INSERT_WITH_COST(τ, Q_g, T, L, d_T, τ_⋆)
44 COMPUTE_COST(τ, Q_g, τ_⋆);
45 T← T∪ τ;
46 if (τ.d_τ==d_T+1) then
47 L.push(τ_new);
```

TABLE I

DIFFUSION PROCESS OF THE PARTIAL MOTION PLANNER

$\mathcal{T}$ which is the closest to the chosen configuration according to a predefined distance metrics and than chooses the nearest feasible command with NEAREST_COMMAND function. The case 3 represents the direct search in the control space of currently kinodynamically feasible commands with RAND_COM_SPACE ($s^3(t_{k+2})$).

For all the three possible expansion methods, the newly obtained nodes/states and the arcs a that connect them to their predecessor nodes in the tree have to be checked for collision against all the dynamic and static obstacles. This is done by discretizing the connecting arc trajectories a in time and verifying possible polygonal intersections resulting from the vehicle and obstacles intermediate configurations in the EXTEND_WITH_COLLISION_CHECK function. Moreover, if the predecessor of an expanded node is at the depth $d_\tau = 0$, additional safety check has to be performed in the function EXTEND_WITH_SAFETY_CHECK (Table I.L32-L40) - the vehicle must be able to come to a full-stop without collision (the state is therefore ICS safe, further details in §VI). In Fig. 4 this is depicted by the example of the break-state $s^3(t_{k+2}+T_b)$ originating from state $s^3(t_{k+2})$, where the collision-check test is performed for the duration of the breaking manoeuvre. The brake time T_b is related to the dynamic capabilities of the vehicle, *ie* max linear deceleration $\dot{v}_{l,decc}$:

$$T_b = \frac{v_l}{\dot{v}_{l,decc}} \tag{8}$$

where the longitudinal velocity depends on each given state, in this case $s^3(t_{k+2})$.

As already mentioned, the diffusion process computation is limited to $t_s \leq T_d$, where at each new computational increment another layer of nodes is added to the tree $\mathcal{T}$. The final depth of $\mathcal{T}$ is not determined a-priori, nor how close the best trajectory will be with respect to the goal configuration. In Fig. 4 the final best trajectory is depicted with $\pi(t_k)$, referring to the navigation cycle when it was computed, whereas the trajectory's final state $\tilde{s}(t_{k+n})$ can be arbitrarily far in the future (n is a free parameter).

Regarding the goal search, *ie* finding a trajectory towards a goal configuration, there are two approaches distinguished here:

1) *waypoint following*: the current set of waypoints $\mathcal{Q}_g$is the next topological node to reach in the environment (see Sec.III), such as an intersection, route crossing, lane changing waypoint, etc.;
2) *route following*: the current set of waypoints $\mathcal{Q}_g$is a collection of intermediate configurations, which together describe a route between topological nodes.

In case 1 the cost function of a node to determine the best trajectory is based on a distance metric $\|\tilde{s} - \{q_g, c_q\}\|$ between the goal waypoint with its constraints and the last state $\tilde{s}$ on a trajectory:

$$w_\tau(\tilde{s}) = \alpha_g \cdot \|\tilde{s} - \{q_g, c_q\}\| + \alpha_t \cdot t_\tau(\tilde{s}) \tag{9}$$

α_g and α_t are the weighting factors between minimizing the distance to the waypoint and minimizing the cumulative time $t_\tau(\tilde{s})$ to reach it, respectively.

In case 2 there is more information available based on the environment structure, in the form of a route. These geometrical configurations can be followed with a type of path following technique, with the possibility of deviating from the route based on the tree structure, if the dynamic obstacles trajectories require such evasive manoeuvres. However, in the

absence of dynamic obstacles $\mathcal{O}_d$ and proper route definition according to the a-priori knowledge of static obstacles $\mathcal{O}_s$, the vehicle should follow the route as close as possible. Assuming that a path following controller (implementation details on the controller can be found in [12], [13]) computes an error function $\mathcal{E}$ which describes the discrepancy between a particular vehicle state s and the route $\mathcal{Q}_g$, then the cost of a node can be computed in the error terms as:

$$w_\tau(\tilde{s}) = \sum_{j=1}^{N_{\tilde{s}}} \|\mathcal{E}(s_j, \mathcal{Q}_g)\| \tag{10}$$

where the set of discretized states $\{s_{j=1} = s(k+1) \dots s_j \dots s_{N_{\tilde{s}}} = \tilde{s}\}$ with the associated arcs a forms a trajectory $\tilde{\pi}$. The cost calculation step is performed in the INSERT_WITH_COST function (Tab. I.L44-L47). The best trajectory $\pi\star = \pi(k)$ which will be applied in the next navigation cycle can be therefore determined from node $\tau_\star$ with minimum overall cost w_τ, iff $\tau_\star \neq \varnothing$.

VI. Safety Issues

The purpose of this section is to explore the safety issues related to the navigation scheme proposed. The diffusion technique presented in §V aims at building a tree embedded in the state×time space of the system $\mathcal{R}$ and to extract from this tree a partial motion π that is used during the next time cycle to drive the system towards its goal.

The concept of Inevitable Collision State (ICS)[4] [14] and the motion safety criteria introduced in [3] show that it does not suffice that each partial motion π be collision-free to ensure the safety of $\mathcal{R}$. From a theoretical point of view, the safety of $\mathcal{R}$ is guaranteed if and only each π is ICS-free up to the time T_d (that corresponds to the initial state of the partial motion that is to be computed at the next navigation cycle), because then, at the next navigation cycle, the navigation module *always* has a safe evasive manoeuvre available.

Now, checking whether a given state of π is an ICS or not requires in theory the *full* knowledge of the environment of $\mathcal{R}$ and its future evolution, *ie* the knowledge of the space-time $\mathcal{W} \times [0, \infty)$. In practice however, one has to deal with the sensors' limited field of views and the elusive nature of the future. Knowledge about the environment of $\mathcal{R}$ is thus limited both *spatially* and *temporally*: it is limited to $\mathcal{W}_p \times [0, T_p]$ where $\mathcal{W}_p \subset \mathcal{W}$ denotes the subset of the environment which is perceived and T_p the prediction horizon. To further ensure safety with respect to the objects that lie outside of $\mathcal{W}_p$, its boundary is treated in a manner similar to [14] or [15] as a potentially moving object whose motion direction is unknown but whose velocity is upper-bounded.

$\mathcal{W}_p \times [0, T_p]$ and the objects within, fixed and moving, yields in the state×time space of $\mathcal{R}$ a set of ICS which is only an approximation of the true set of ICS generated by $\mathcal{W} \times [0, T_p]$. This is the very reason why it is impossible to guarantee an absolute level of safety (absolute in the sense that it can be guaranteed that $\mathcal{R}$ will never end up in an ICS and therefore crash eventually). This intrinsic impossibility compels us to settle for weaker levels of safety. Although weaker, the important thing is that such levels of safety will be guaranteed given the information that $\mathcal{R}$ knows about its environment, *ie* given $\mathcal{W}_p \times [0, T_p]$. We have explored two different levels of safety, they are detailed in the next two sections.

[4]A state is an ICS iff a collision eventually occurs no matter how $\mathcal{R}$ moves.

A. Safety Level #1

The first safety level we have seeked to enforce is the simplest one maybe. It guarantees that, should a collision ever occur, $\mathcal{R}$ will be at rest. In other words, if a collision is inevitable, it can be guaranteed that $\mathcal{R}$ always have the possibility to brake down and stop before the collision occurs. Such a safety level is a form of *passive* safety in the sense that $\mathcal{R}$ will never actively collide with an object. It is henceforth called *Passive Safety* and denoted by PS.

Under PS, a state s is considered as being safe iff there exists at least one braking manoeuvre starting at s which is collision-free until the time where $\mathcal{R}$ has stopped. PS yields the following definition for a safe state:

*Def. 1 (**Passive Safety**):* a state s is safe under PS (or **p-safe**) iff there exists at least one braking manoeuvre starting at s and collision-free until T_b, with T_b the time where $\mathcal{R}$ is at rest (the *braking time*).

In practice, the function EXTEND.WITH.SAFETY.-CHECK (*cf* Table I) samples a finite and discrete set of braking manoeuvres and checks them for collision against $\mathcal{W}_p \times [0, T_p]$. If one collision-free manoeuvre exists the state considered is labeled as p-safe and unsafe otherwise.

B. Safety Level #2

In a way, PS leaves most of the collision-avoidance burden to the other objects. In certain situations however, this is unsatisfactory: $\mathcal{R}$ may for instance decide to move on a railway track to reach its goal because, under PS, it is safe to do so (indeed $\mathcal{R}$ would have the time to stop before being hit by the train). Unfortunately, the train in spite of its best efforts may not be able to avoid crashing into $\mathcal{R}$ because of its own dynamics.

In an environment where the moving objects are assumed to be *friendly*, *ie* seeking to avoid collisions, and for which a certain knowledge about their dynamic properties is available, it can be desirable to enforce a stronger level of safety. This second safety level guarantees that, should a collision ever occur, $\mathcal{R}$ will be at rest and the colliding object would have had the time to slow down and stop before the collision had it wanted to. This safety level is henceforth called *Passive Friendly Safety* and is denoted PFS. It yields the following definition for a safe state:

*Def. 2 (**Passive Friendly Safety**):* a state s is safe under PFS (or **pf-safe**) iff there exists at least one braking manoeuvre starting at s and collision-free until $T_b + T_{ob}$, with T_b the braking time of $\mathcal{R}$, and T_{ob} the maximum braking time of the moving objects present in the environment.

The conservative nature of Def. 2 should be noted. It is possible in practice to refine it in order for example to take into account the dynamics of the particular moving objects that would collide with $\mathcal{R}$ when it follows a particular braking manoeuvre. For the time being, Def. 2 is left as is.

Other safety levels could be proposed. The ultimate one of course is to determine safety with respect to the set of ICS which is defined by $\mathcal{W}_p \times [0, T_p]$. Given the complexity of characterizing this ICS set, Passive Safety and Passive Friendly Safety constitutes interesting alternatives in the sense that they can be computed efficiently and provide an adequate level of safety.

VII. Experimental platform

A. SmartTer

The SmartTer is a passenger car equipped for autonomous driving. The system has a localization module based on sensor fusion, in the line of that described in [16]. To accurately localize the vehicle, four different sensors are used: DGPS, IMU, optical gyro and vehicle sensors (wheel encoders and steering angle sensor). The combination of their measurements allows the estimation of the vehicle's 6 degrees of freedom *eg* the 3D position (x, y, z) and the attitude (roll, pitch, heading). The details of the localization system used are described in [17].

Fig. 5. The SmartTer passenger car equipped for autonomous driving. Behind the windscreen we have the camera system, on the sides of the roof we have the tilted laser scanners for corner protection. Currently, the front Sick has been replaced by an Alasca Scanner.

For environment detection we use one IBEO Alasca XT laser scanner mounted at the front of the vehicle and two Sick LMS 291 mounted on the roof to protect the vehicle corners as well as a Sony Camera for vision.

B. Simulator

In order to facilitate system integration and to perform tests where the ground truth is available, we have performed our experiments using the Morsel simulator. Morsel has been developed in our laboratory and is based on the Panda3d engine [18]. Morsel features emulation for the two lower level layers of fig. 1. This includes cameras, range sensors, and standard cinematic models such as Ackermann, differential, etc.

The simulator can be easily extended using Python or C++. Another interesting feature is its ability run in "real" or "virtual" time modes. In the former mode, the simulator will try to produce sensor output at the real frequency; in the second mode time is emulated by putting world events in a priority queue according to their frequency. Since our system uses the Carmen [19] library for communications, low-level sensor and actuator driver modules are effectively decoupled from high-level algorithm. This makes it possible to plug the simulator to those algorithms in a transparent fashion.

VIII. Simulation results

Simulation results are presented in a generic environment, where no specific structure is present (*eg* parking lot), therefore, the navigation scheme based on waypoint following is applicable here (see §V, case 1) for goal search. The dynamic obstacles present in the Scene 1 (Fig. 6) and 2 (Fig. 8) are pedestrians and static and moving vehicles. The estimated obstacle positions and their future trajectories are depicted in navigation snapshots in Fig.7 and Fig. 9, with the currently active waypoint is drawn as a circular region with a specified orientation (according to the given constraints). The trajectory diffusion starting from the ego-vehicle contains tree nodes that are free of obstacles (green) and prohibited nodes (marked red), whereas the currently best trajectory towards the waypoint is marked in magenta. Note that the given navigation snapshots are based on a certain time instant $t_k$and that the prohibited regions depend on all the future motions of the obstacles. It can be seen from the results that the ego-vehicle is both able to reach the waypoint with the constraints included (Fig. 7), while negotiating moving obstacles, even preventing head-on collisions (Fig. 9).

IX. Conclusion and Outlook

This paper has presented the deliberative part of the navigation architecture for the SmartTer platform, comprising two main component: (a) *route planning*, which finds a set of configurations between two given points of the environment while taking into account given traffic rules; and (b) *partial motion planning*, which handles the actual execution of the plan while taking into account the dynamic elements of the world. A key aspect of the navigation architecture proposed is that a special attention is paid to the motion safety issue, *ie* the ability to avoid collisions. Different safety levels

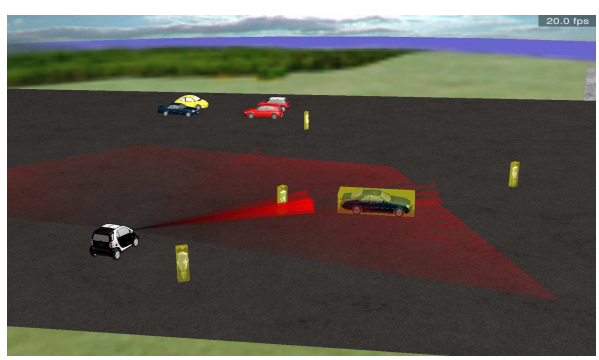

Fig. 6. World view (scene1)

are explored and their operational conditions are explicitly spelled out (something which is usually not done). The results depict safe navigation in dynamic scenarios which include both moving vehicles and pedestrians, where the architecture has been implemented in both real and simulated platforms, although only simulation results are provided at the time of paper writing.

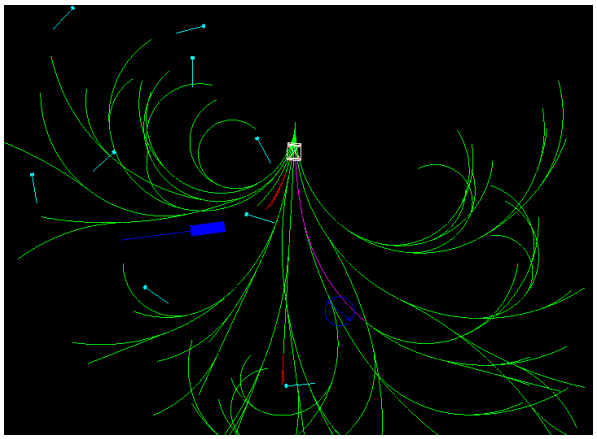

Fig. 7. Navigation output (scene1)

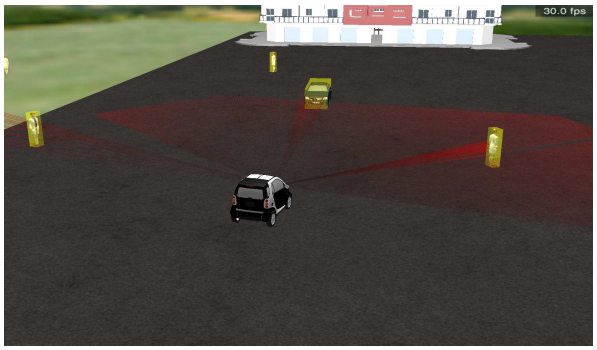

Fig. 8. World view (scene2)

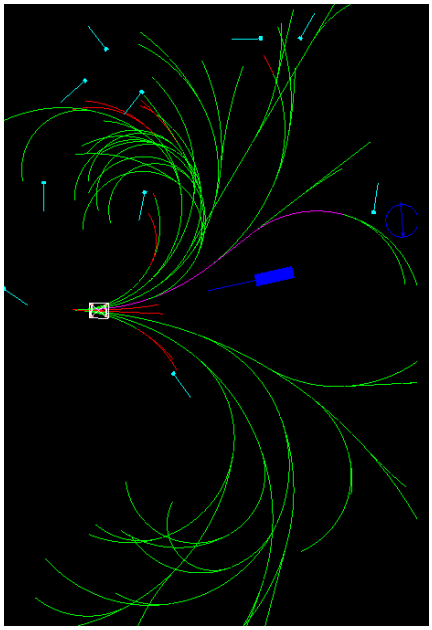

Fig. 9. Navigation output (scene2)

On the theoretical side, an interesting research direction is the exploration of more advanced motion prediction techniques in order to improve both the accuracy and the time horizon of our safety checks.

X. Acknowledgments

This work has partly been supported by the EC under contract number FP6-IST-027140-BACS, as well as by the European project URUS (Ubiquitous Networking Robots in Urban Settings).

References

[1] N. J. Nilsson, "Shakey the robot," Technical note 323, AI Center, SRI International, Menlo Park, CA (US), Apr. 1984.

[2] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, MIT Press, 2004.

[3] Th. Fraichard, "A short paper about motion safety," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Roma (IT), Apr. 2007.

[4] J. Borenstein and Y. Korem, "The vector field histogram — fast obstacle avoidance for mobile robts," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 3, June 1991.

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

[6] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, Mar. 1997.

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

[8] S. Petti and Th. Fraichard, "Partial motion planning framework for reactive planning within dynamic environments," in *Proc. of the IFAC/AAAI Int. Conf. on Informatics in Control, Automation and Robotics*, Barcelona (SP), Sept. 2005.

[9] K. Macek, M. Becker, and R. Siegwart, "Motion planning for car-like vehicles in dynamic urban scenarios," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[10] "Route network definition file (RNDF) and mission data file (MDF) formats," Mar. 2007, http://www.darpa.mil/grandchallenge/docs.

[11] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *Int. Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[12] R. Solea and U. Nunes, "Trajectory planning with velocity planner for fully-automated passenger vehicles," in *Proc. of the IEEE Intelligent Transportation Systems Conference (ITSC)*, 2006.

[13] K. Macek, R. Philippsen, and R. Siegwart, "Path following for autonomous vehicle navigation with inherent safety and dynamics margin," in *Proc. of the IEEE Int. Vehicles Symposium (IV)*, 2008.

[14] Th. Fraichard and H. Asama, "Inevitable collision states. a step towards safer robots?," *Advanced Robotics*, vol. 18, no. 10, 2004.

[15] R. Alami, T. Simeon, and K. Madhava Krishna, "On the influence of sensor capacities and environment dynamics onto collision-free motion plans," in *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Lausanne (CH), Oct. 02.

[16] G Dissanayake, S Sukkarieh, E. Nebot, and H. Durrant-Whyte, "The aiding of a low-cost strapdown inertial measurement unit using vehicle model constraints for land vehicle applications," *IEEE Transactions on Robotics and Automation*, 2001.

[17] P. Lamon, S. Kolski, and R. Siegwart, "The SmartTer - a vehicle for fully autonomous navigation and mapping in outdoor environments," in *Proc. of the CLAWAR*, Brussels, Belgium, 2006.

[18] Mike Goslin and Mark R. Mine, "The panda3d graphics engine," *Computer*, vol. 37, no. 10, pp. 112–114, 2004.

[19] M. Montemerlo, N. Roy, and S. Thrun, "Carmen: Carnegie mellon robot navigation toolkit," http://www.cs.cmu.edu/~carmen/.