Int. J. Appl. Math. Comput. Sci., 2009, Vol. 19, No. 3, 413–424
DOI: 10.2478/v10006-009-0034-2

amcs

RELIABLE ROBUST PATH PLANNING WITH APPLICATION TO MOBILE ROBOTS

ROMAIN PEPY, MICHEL KIEFFER, ERIC WALTER

L2S-CNRS-SUPELEC—Univ. Paris-Sud
3 rue Joliot-Curie, 91192 Gif-sur-Yvette, France
e-mail: romain.pepy@onera.fr,{kieffer,walter}@lss.supelec.fr

This paper is devoted to path planning when the safety of the system considered has to be guaranteed in the presence of bounded uncertainty affecting its model. A new path planner addresses this problem by combining Rapidly-exploring Random Trees (RRT) and a set representation of uncertain states. An idealized algorithm is presented first, before a description of one of its possible implementations, where compact sets are wrapped into boxes. The resulting path planner is then used for nonholonomic path planning in robotics.

Keywords: interval analysis, path planning, robust control, state-space models.

1. Introduction

Consider a system described by a continuous-time state-space model. Designing some control input to drive this system from a possibly uncertain initial state to a desired final state is a well-known robust control problem (Ackermann *et al*., 1993; Francis and Khargonekar, 1995). This problem is made more complicated when constraints on the control input and on the evolution of the state also have to be satisfied. To solve it, a model of the system is usually assumed to be available, where noise variables account for the fact that this model is only an approximation of reality. The control input then has to be chosen in such a way that the system reaches the desired final state, despite uncertainty in the initial state and the presence of noise, i.e., the control input has to be *robust* to any type of uncertainty.

This paper focuses on applications in robotics, where the robust control problem becomes a reliable path-planning problem (Latombe, 1991). Consider, for example, a vehicle moving in a two-dimensionnal structured environment. This vehicle should be driven from an initial state or *configuration* (position and orientation of the vehicle with respect to a frame attached to the environment) to a final desired configuration, despite the presence of uncertainty related to the model of the vehicle, to imperfect embedded sensors, to approximately charted obstacles, etc. The control input and the corresponding paths (succession of states) achieving this goal without collision are said to be safe or *reliable*.

Path planners involving Rapidly-exploring Random Trees (RRT) (LaValle, 1998; LaValle and Kuffner, 2001a; LaValle and Kuffner, 2001b) represent the state-of-the-art in random search. They allow an efficient exploration of the configuration space but, to the best of our knowledge, do not provide any robustness to model uncertainty. When taken into account, configuration uncertainty is usually described probabilistically, e.g., by a multivariate Gaussian probability density function (Lambert and Gruyer, 2003; Gonzalez and Stentz, 2005; Pepy and Lambert, 2006). The main drawback of path planners based on this description is that the reliability of the path obtained may be guaranteed at best up to a given confidence level.

To facilitate path planning in the presence of uncertainty, information allowing the vehicle to localize itself is sometimes assumed to be available. In (Lazanas and Latombe, 1995; Bouilly *et al.,* 1995; Fraichard and Mermond, 1998; Gonzalez and Stentz, 2004; 2007), for example, relocalization zones in which the configurations become perfectly (or at least much more accurately) known are considered. This technique is rather efficient but requires the preparation of these relocalization zones. In (Lambert and Gruyer, 2003; Pepy and Lambert, 2006), a complex model of exteroceptive sensors (sonars) and an extended Kalman filter are used. To provide distance measurements during path planning, sonars are simulated assuming that the vehicle is located at the mean of the mul-

tivariate Gaussian function that characterizes location uncertainty. The resulting simulated measurements are then used to reduce uncertainty. If this technique facilitates the calculation of a path, it of course does not allow any statement about the reliability of this path.

This paper presents a first conceptual reliable robust path planner, assuming that all uncertain quantities are *bounded* with *known bounds*. At each time instant, uncertain configurations are represented by possibly non-connected sets. The proposed path planner takes advantage of the ability of RRTs to explore the whole configuration space efficiently. Starting from some uncertain initial configuration (represented by a set), the planner aims at driving the vehicle to a final configuration *set* (it will not be possible to drive it accurately to a point final configuration). Provided that the assumptions on the error bounds are not violated, if a robust path is found using this new path planner, its reliability will be *guaranteed*.

This paper is organized as follows: In Section 2, the two types of robust path planning problems to be addressed are presented. The principle of path planners based on RRTs is described in Section 3. Section 4 provides a conceptual extension of these planners to sets and Box-RRT, one of its implementable counterparts where these sets are represented by boxes (or interval vectors). Section 5 applies Box-RRT to path planning for non-holonomic vehicles. Examples of path planning tasks for a vehicle are given in Section 6, before some conclusions.

2. Reliable robust path planning

Consider a system whose evolution is described by the continuous-time state equation

$$\frac{\mathrm{d}s\left(t\right)}{\mathrm{d}t} = f(s\left(t\right), u\left(t\right), w\left(t\right)), \tag{1}$$

where $s\left(t\right) \in \mathbb{S} \subset \mathbb{R}^n$ is the state of the system, u is some bounded input function with values in $[u]$ and w is some random bounded state perturbation function remaining in $[w]$. It is assumed that u belongs to $\mathcal{U}_{[u]}^{\Delta t}$, the set of piecewise-constant bounded functions over intervals of the form $[k\Delta t, (k+1)\,\Delta t[$, with $\Delta t > 0$ and $k \in \mathbb{N}$, and that w belongs to $\mathcal{W}_{[w]}$, the set of functions bounded in $[w]$. For all $t \in [k\Delta T, (k+1)\,\Delta T[$, $u \in \mathcal{U}_{[u]}^{\Delta t}$, and $w \in \mathcal{W}_{[w]}$, $g\left(s,t\right) = f\left(s, u\left(t\right), w\left(t\right)\right)$ is assumed ℓ-Lipschitz over $\mathbb{S}$.

The state-space $\mathbb{S}$ is partitioned into $\mathbb{S}_{\text{free}}$, to which the state of the system is allowed to belong, and $\mathbb{S}_{\text{obs}} = \mathbb{S} \setminus \mathbb{S}_{\text{free}}$, to which it is not. $\mathbb{S}_{\text{obs}}$ represents the results of constraints imposed on the system, e.g., by its environment.

At time $t = 0$, $s\left(0\right)$ is assumed to belong to some known set $\mathbb{S}\left(0\right) = \mathbb{S}_{\text{init}} \subset \mathbb{S}_{\text{free}}$. The system has to be driven to a given set of goal states $\mathbb{S}_{\text{goal}} \subset \mathbb{S}_{\text{free}}$. The aim of *robust* path planning is then to design an input function $u \in \mathcal{U}_{[u]}^{\Delta t}$ such that the system reaches $\mathbb{S}_{\text{goal}}$, without entering $\mathbb{S}_{\text{obs}}$ at any time instant, whatever the initial state $s \in \mathbb{S}_{\text{init}}$ and the noise function $w \in \mathcal{W}_{[w]}$. A planned path is *reliable* when a given function $u \in \mathcal{U}_{[u]}^{\Delta t}$ can be *proved* to drive the system from any $s \in \mathbb{S}_{\text{init}}$ to a final state in $\mathbb{S}_{\text{goal}}$.

As will be seen below, there may be several formulations of this robust path planning problem.

2.1. Problem 1: Path planning. A first formulation of the robust path planning problem amounts to determining whether

$$\begin{aligned}&\exists K > 0 \text{ and } \exists u \in \mathcal{U}_{[u]}^{\Delta t} \text{ such that}\\ &\forall s \in \mathbb{S}_{\text{init}} \text{ and } \forall w \in \mathcal{W}_{[w]},\, s\left(K\Delta t\right) \in \mathbb{S}_{\text{goal}} \text{ and}\\ &\forall t \in [0, K\Delta t]\,,\, s\left(t\right) \in \mathbb{S}_{\text{free}},\end{aligned} \tag{2}$$

where $s\left(t\right)$ is the solution of (1).

In (2), the *same* sequence of inputs has to drive the system robustly from its imprecisely known initial state to a final state belonging to $\mathbb{S}_{\text{goal}}$. If the initial uncertainty on the state or the state perturbation is too large, or if $\mathbb{S}_{\text{free}}$ has a complex structure and the distance between $\mathbb{S}_{\text{init}}$ and $\mathbb{S}_{\text{goal}}$ is too long, it may become quite difficult to find such a sequence of inputs. It may then be convenient to relax Problem 1 and consider Problem 2, presented in the next section, instead.

2.2. Problem 2: Reachability analysis. Even if a solution to (2) exists, actual control inputs are usually not applied in open loop. Instead, an observer is used to estimate the state evolution using measurements provided by sensors, see, e.g., (Luenberger, 1966). With this improved knowledge, it may be very useful to update path planning from time to time. In such a context, determining whether there exists a unique sequence of inputs that drives the system to $\mathbb{S}_{\text{goal}}$ whatever the initial state in $\mathbb{S}_{\text{init}}$ is too stringent. It suffices to know whether for any initial state $s \in \mathbb{S}_{\text{init}}$ there exists a sequence of inputs that drives the system from s to $\mathbb{S}_{\text{goal}}$. This is typically a *reachability* problem: One has to determine whether $\mathbb{S}_{\text{goal}}$ is reachable from any state in $\mathbb{S}_{\text{init}}$ and for any $w \in \mathcal{W}_{[w]}$.

Formally, one has to determine whether

$$\begin{aligned}&\forall s \in \mathbb{S}_{\text{init}}, \exists K > 0 \text{ and } \exists u \in \mathcal{U}_{[u]}^{\Delta t} \text{ such that}\\ &\forall w \in \mathcal{W}_{[w]},\, s\left(K\Delta t\right) \in \mathbb{S}_{\text{goal}} \text{ and}\\ &\forall t \in [0, K\Delta t]\,,\, s\left(t\right) \in \mathbb{S}_{\text{free}},\end{aligned} \tag{3}$$

where $s\left(t\right)$ is again the solution of (1).

3. Rapidly-exploring Random Trees (RRT)

As for several non-reliable path planning algorithms, the RRT algorithm will be the corner-stone of the proposed

Algorithm 1 RRT($s_{\text{init}} \in \mathbb{S}_{\text{free}}$, $\mathbb{S}_{\text{goal}} \subset \mathbb{S}_{\text{free}}$, $\Delta t \in \mathbb{R}^+$, $K \in \mathbb{N}$)

1: G.init(s_{init})
2: $i = 0$
3: **repeat**
4: $s_{\text{rand}} \leftarrow$ random_vector($\mathbb{S}_{\text{free}}$)
5: $s_{\text{new}} \leftarrow$ RRT_extend(G, s_{rand}, Δt)
6: **until** i++> K **or** ($s_{\text{new}} \neq$**null and** $s_{\text{new}} \in \mathbb{S}_{\text{goal}}$)
7: **return** G

Algorithm 2 RRT_extend(G, s_{rand}, Δt)

1: $s_{\text{near}} \leftarrow$ nearest_neighbor(G, s_{rand})
2: $u \leftarrow$ select_input(s_{rand}, s_{near})
3: $s_{\text{new}} \leftarrow$ new_state(s_{near}, u, Δt)
4: **if** collision_free_path(s_{near},s_{new},u,Δt) **then**
5: G.add_node(s_{new})
6: G.add_edge(s_{near}, s_{new}, u)
7: **return** s_{new}
8: **end if**
9: **return null**

Algorithm 3 Set-RRT($\mathbb{S}_{\text{init}} \subseteq \mathbb{S}_{\text{free}}$, $\mathbb{S}_{\text{goal}} \subseteq \mathbb{S}_{\text{free}}$, $\Delta t \in \mathbb{R}^+$, $K \in \mathbb{N}$)

1: G.init($\mathbb{X}_{\text{init}}$)
2: $i \leftarrow 0$
3: **repeat**
4: $\mathbb{S}_{\text{rand}} \leftarrow$ random_set($\mathbb{S}_{\text{free}}$)
5: $\mathbb{S}_{\text{new}} \leftarrow$ Set-RRT_extend(G, $\mathbb{S}_{\text{rand}}$, Δt)
6: **until** i++> K **or** ($\mathbb{S}_{\text{new}} \neq$**null and** $\mathbb{S}_{\text{new}} \subset \mathbb{S}_{\text{goal}}$)
7: **return** G

Algorithm 4 Set-RRT_extend(G, $\mathbb{S}_{\text{rand}}$, Δt)

1: $\mathbb{S}_{\text{near}} \leftarrow$ nearest_neighbor(G, $\mathbb{S}_{\text{rand}}$)
2: $u \leftarrow$ select_input($\mathbb{S}_{\text{rand}}$, $\mathbb{S}_{\text{near}}$)
3: $\mathbb{S}_{\text{new}} \leftarrow$ prediction($\mathbb{S}_{\text{near}}$, u, Δt)
4: **if** collision_free_path($\mathbb{S}_{\text{near}}$,$\mathbb{S}_{\text{new}}$,u,Δt) **then**
5: G.add_guaranteed_node($\mathbb{S}_{\text{new}}$)
6: G.add_guaranteed_edge($\mathbb{S}_{\text{near}}$, $\mathbb{S}_{\text{new}}$, u)
7: **return** $\mathbb{S}_{\text{new}}$
8: **end if**
9: **return null**

reliable and robust path planner. The structure and properties of the RRT algorithm are thus now briefly recalled. In the remainder of this section, it is assumed that the initial state $s\,(0) = s_{\text{init}}$ is perfectly known, and that no perturbation affects the state equation (1).

3.1. Description. The RRT algorithm (Kuffner and LaValle, 2000; LaValle and Kuffner, 2001b) is an incremental method aimed at quickly exploring a given configuration space from a given starting configuration. It is described in Algorithms 1 and 2. First, the tree G is initialised with a single node corresponding to s_{init}. Then, a state $s_{\text{rand}} \in \mathbb{S}_{\text{free}}$ is chosen at random. The `nearest_neighbor` function searches in the tree G for the node s_{near} that is closest to s_{rand} according to some metric d. A control input $u \in [u]$ is then chosen (for instance, at random). Integrating (1) over a time interval Δt with the initial condition s_{near} and a constant control input u results in a new state s_{new}. If it can be *proved* that all state values along the trajectory between s_{near} and s_{new} lie in $\mathbb{S}_{\text{free}}$, then the trajectory between s_{near} and s_{new} is *reliable* and s_{new} is added to G and connected to s_{near}. Otherwise, s_{new} is not added to G. A new random state is chosen to start the next iteration of the algorithm. A path is found when $s_{\text{new}} = s_{\text{goal}}$, or (more realistically) when $s_{\text{new}} \in \mathbb{S}_{\text{goal}}$.

Figure 1 illustrates the growth of the tree G with the number of iterations of the RRT algorithm when $\mathbb{S} = [0, 100]^2$, $\dot{s} = u$, with $s \in \mathbb{S} \subset \mathbb{R}^2$, $u \in [0, 1]^2$ and $\Delta t = 100$ ms.

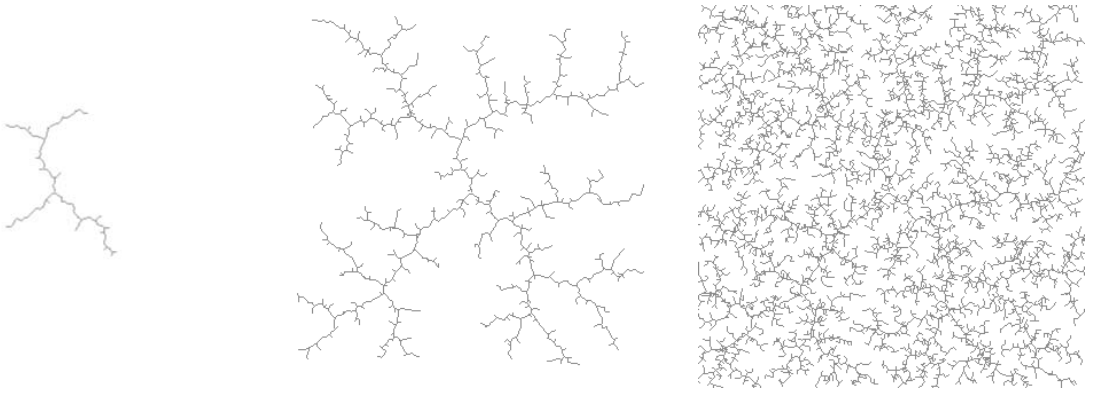
3.2. Improvements. Much attention has been dedicated to improving RRT. In (LaValle and Kuffner, 2001b), the generation of s_{rand} is modified by biasing the tree toward s_{goal}, which increases the planning speed for some specific $\mathbb{S}_{\text{free}}$. Instead of choosing s_{rand} in the whole $\mathbb{S}_{\text{free}}$, another option is to choose it with a probability $p > 0$ in a given subset $\mathbb{S}_{\text{rand}}$ of $\mathbb{S}_{\text{free}}$. If $\mathbb{S}_{\text{rand}} = \{s_{\text{goal}}\}$, one obtains the RRT-*Goalbias* algorithm, and if $\mathbb{S}_{\text{rand}}$ is the circle centered on s_{goal} with a radius $\min_{s \in G} d(s, s_{\text{goal}})$, one gets the RRT-*GoalZoom* algorithm.

4. Set-RRT and Box-RRT

In order to cope with an uncertain initial configuration and bounded state pertubations, the classical RRT path planner has to be adapted to deal with sets. The first part of this section is devoted to the presentation of a new conceptual algorithm, before describing one of its implementable counterparts.

4.1. Set-RRT. Set-RRT aims at generating a graph G consisting of nodes associated with *sets* in state space. The structure of Set-RRT is very close to that of the classical RRT algorithm, where nodes are associated with vectors. The main changes concern the metric required to evaluate distances between sets, the prediction function, which has to determine the evolution of uncertain states according to (1), and the collision test to determine whether all possible trajectories between two consecutive sets are reliable. The principle of Set-RRT is given in Algorithms 3 and 4.

At Step 4, $\mathbb{S}_{\text{rand}}$ is most often chosen as a point vector, but making it a set allows replacement of $\mathbb{S}_{\text{rand}}$ by $\mathbb{S}_{\text{goal}}$ for the implementation of set variants of *Goalbias* and *GoalZoom*. Set-RRT stops when either the number of nodes



(a) (b) (c)

Fig. 1. Growth of the tree built by the RRT algorithm: (a) 100 nodes, (b) 600 nodes, (c) 6000 nodes.

generated reaches its limit K, or when the goal area is reached, *i.e.*, the tree includes a node associated with a set $\mathbb{S}_k$ such that $\mathbb{S}_k \subset \mathbb{S}_{\text{goal}}$.

4.2. Box-RRT. Dealing with general sets of $\mathbb{R}^n$ is very difficult, even for the simplest uncertain state equations. Wrappers (Jaulin *et al*., 2001) guaranteed to contain the sets $\mathbb{S}_k$ have to be used to get an implementable counterpart to Set-RRT. Candidate wrappers are, for example, ellipsoids (Schweppe, 1973), zonotopes (Alamo *et al*., 2003), interval vectors (Moore, 1979) or a union of interval vectors (Kieffer *et al.*, 2001; 2002). In what follows, interval vectors, or *boxes*, are used to represent uncertain states. These are quite simple sets, which may provide a very coarse description of complex-shaped sets. Using more accurate wrappers may increase the number of problems to which solutions may be found.

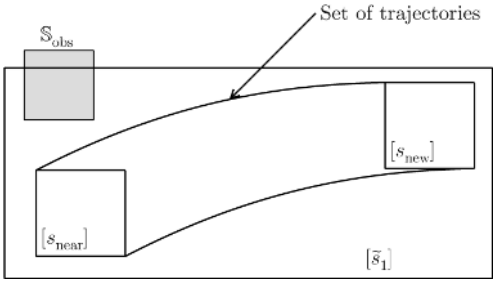


Fig. 2. Set of trajectories between $[s_{\text{near}}]$ and $[s_{\text{new}}]$, wrapped in $[\tilde{s}_1]$, is reliable, but this cannot be proved, since $[\tilde{s}_1]$ has a non-empty intersection with $\mathbb{S}_{\text{obs}}$.

In what follows, a specialization of Algorithm 3 to boxes is called Box-RRT. In Box-RRT, the Hausdorff distance (Berger, 1987) between boxes may be used by the `nearest_neighbor` function. The box $[s_{\text{new}}] = [s_{k+1}]$ containing all possible state values at time $(k+1)\,\Delta t$ considering that the state is in $[s_{\text{near}}]$ at time $k\Delta t$ and that the input $u_k \in [u]$ is constant over $[k\Delta t, (k+1)\,\Delta t[$ must be computed while taking into account the bounded state perturbation. This may be performed by a set `prediction` function involving guaranteed numerical integration, as proposed, e.g., in (Jaulin, 2002; Kieffer and Walter, 2003; 2006; Raissi *et al.*, 2004). Finally, the set collision test that guarantees the reliability of every path between $[s_{\text{near}}]$ and $[s_{\text{new}}]$ implemented in `collision_free_path` requires to wrap all possible state trajectories between $[s_{\text{near}}]$ and $[s_{\text{new}}]$. This is again performed using guaranteed numerical integration. Note that wrapping may be so coarse that a path may not be deemed robustly reliable even if it actually is, see Fig. 2. On the contrary, in situations such as that of Fig. 3, the set of paths between $[s_{\text{near}}]$ and $[s_{\text{new}}]$ can be easily proved to be robustly reliable.

Figures 4(a) and 4(b) show two paths planned for a system described by the two-dimensionnal uncertain state equation

$$\dot{s} = \frac{1}{1-w}u, \tag{4}$$

where $s \in \mathbb{R}^2$, $w \in [-0.02, 0.02]$, $s_{\text{init}} \in [90, 90.1]^2$, $s_{\text{goal}} = [10, 20]^2$, $u \in [0, 1]^2$, and $\Delta t = 100$ ms. Figure 4(b) illustrates the performance of the *Goalbias* variant of the Box-RRT algorithm with $p = 0.1$.

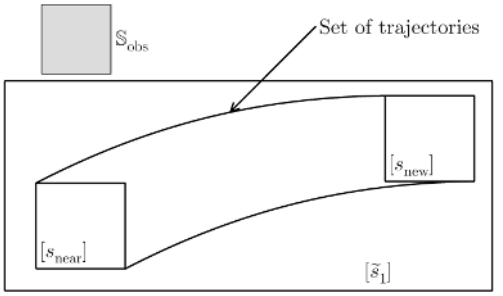
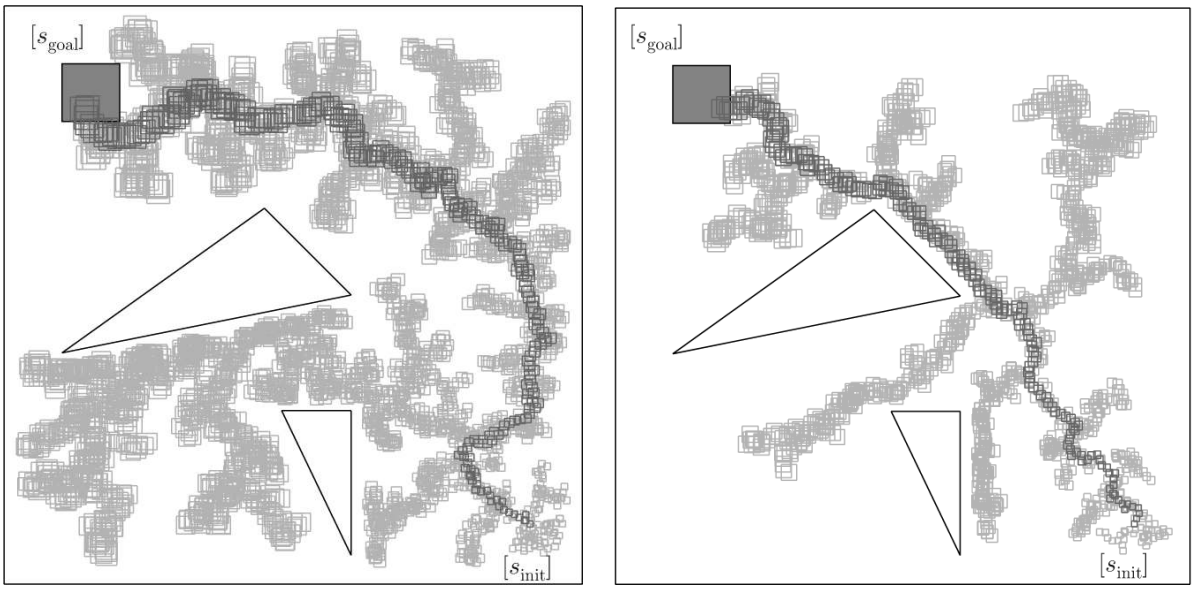


Fig. 3. Set of trajectories between $[s_{\text{near}}]$ and $[s_{\text{new}}]$, wrapped in $[\tilde{s}_1]$, is proved to be reliable, since $[\tilde{s}_1]$ has an empty intersection with $\mathbb{S}_{\text{obs}}$.



(a) (b)

Fig. 4. Paths planned using the Box-RRT algorithm: (a) path planned using the Box-RRT algorithm, (b) path planned using the *Goalbias* variant of Box-RRT.

These first two examples show the ability of Box-RRT to find a reliable path in a simple environment, accounting for uncertainty in the model of the system. Nevertheless, uncertainty is growing along the path, since no measurement is used to reduce it. The next section is devoted to a solution of Problem 2 described in Section 2.

4.3. Reach-RRT: Box-RRT and reachability analysis. In Box-RRT a unique series of constant control inputs over time intervals of width Δt is used to compute $[s_{\text{new}}]$ from $[s_{\text{near}}]$ satisfying (1). They are the same for all $s \in [s_{\text{near}}]$, which is natural for path planning, since the succession of values taken by the control input is important to actually drive the system from $[s_{\text{init}}]$ to $[s_{\text{goal}}]$.

If Box-RRT does not manage to find a unique input function $u \in \mathcal{U}_{[u]}^{\Delta t}$ to solve Problem 1, one may first try to split $[s_{\text{init}}]$ into subboxes and to apply Box-RRT to each of them. A solution to Problem 2 is then obtained because the control input sequence is usually no longer the same for all $s \in [s_{\text{init}}]$. The main difficulty with this technique is that the number of boxes in which $[s_{\text{init}}]$ has to be split so that Box-RRT provides a solution for each of them may be difficult to determine *a priori*. Moreover, instead of getting a single tree, one obtains as many trees as subboxes in $[s_{\text{init}}]$.

We propose instead to generate a single tree, leading to a set of trajectories without branching leading from $[s_{\text{init}}]$ to $[s_{\text{goal}}]$. Between two consecutive boxes $[s_k]$ and $[s_{k+1}]$ of this set of trajectories, the control input may be adapted to each $s \in [s_k]$ to ensure that the system actually reaches $[s_{k+1}]$. This allows a reduction of the size of $[s_{\text{new}}]$ at each iteration with a simple modification of Box-RRT, entitled BoxReduction, executed just after Step 4 of the `extend` function of the Box-RRT algorithm. Algorithms 5 to 8 describe the proposed Reach-RRT algorithm.

Algorithm 5 Reach-RRT($[s_{\text{init}}] \subseteq \mathbb{S}_{\text{free}}$, $[s_{\text{goal}}] \subseteq \mathbb{S}_{\text{free}}$, $\Delta t \in \mathbb{R}^+$, $K \in \mathbb{N}$, $J \in \mathbb{N}$)

1: G.init($[s_{\text{init}}]$)
2: $i \leftarrow 0$
3: **repeat**
4: $\quad [s_{\text{rand}}] \leftarrow$ random_box($\mathbb{S}_{\text{free}}$)
5: $\quad [s_{\text{new}}] \leftarrow$ Reach-RRT_extend(G, $[s_{\text{rand}}]$, Δt, J)
6: **until** i++> K **or** ($[s_{\text{new}}] \neq \emptyset$ **and** $[s_{\text{new}}] \in [s_{\text{goal}}]$)
7: **return** G

Algorithm 6 Reach-RRT_extend(G, $[s_{\text{rand}}]$, Δt, J)

1: $[s_{\text{near}}] \leftarrow$ nearest_neighbor(G, $[s_{\text{rand}}]$)
2: $u \leftarrow$ select_input($[s_{\text{rand}}]$, $[s_{\text{near}}]$)
3: $[s_{\text{new}}] \leftarrow$ prediction($[s_{\text{near}}]$, u, Δt)
4: **if** collision_free_path($[s_{\text{near}}]$,$[s_{\text{new}}]$,u,Δt) **then**
5: $\quad [s_{\text{new}}] \leftarrow$ box_reduction($[s_{\text{near}}]$, $[s_{\text{new}}]$, J)
6: $\quad G$.add_guaranteed_node($[s_{\text{new}}]$)
7: $\quad G$.add_guaranteed_edge($[s_{\text{near}}]$, $[s_{\text{new}}]$, u)
8: $\quad$ **return** $[s_{\text{new}}]$
9: **end if**
10: **return** $\emptyset$

In the `Set-RRT_extend` function of Algorithm 4, assume that $[s_{\text{near}}]$ corresponds to time $k\Delta t$. After Step 4 of this function, one gets $[s_{\text{new}}]$ at $(k+1)\,\Delta t$ corresponding to the set of states consistent with $[s_{\text{near}}]$, the chosen

Algorithm 7 box_reduction($[s_{\text{near}}]$, $[s_{\text{new}}]$, J)

$[s_{\text{toreturn}}] \leftarrow [s_{\text{new}}]$
$\mathcal{S} \leftarrow \text{cut}([s_{\text{near}}], J)$ $\{\mathcal{S} = \{[s_{\text{near}}]_1, [s_{\text{near}}]_2, \ldots, [s_{\text{near}}]_J\}\}$
repeat
  $[s_{\text{red}}] \leftarrow \text{reduce}([s_{\text{toreturn}}])$
  for all $[s_{\text{near}}]_j \in \mathcal{S}$ **do**
    isReduced $\leftarrow$ find_input($[s_{\text{near}}]_j$, $[s_{\text{red}}]$)
    if (isReduced == FAILURE) **then**
      return $[s_{\text{toreturn}}]$
    end if
  end for
  $[s_{\text{toreturn}}] \leftarrow [s_{\text{red}}]$
until TRUE

control input u, and the noise $w \in \mathcal{W}_{[w]}$. The aim is to find some $[s'_{\text{new}}] \subset [s_{\text{new}}]$, with minimum width, such that $\forall s \in [s_{\text{near}}]$, $\exists u_k \in [u]$ satisfying

$$\forall w \in \mathcal{W}_{[w]}, s\left((k+1)\,\Delta t\right) \in [s'_{\text{new}}] \text{ and } \forall t \in [k\Delta t, (k+1)\,\Delta t], s\,(t) \in \mathbb{S}_{\text{free}}. \quad (5)$$

This problem may be quite difficult to solve. The following sub-optimal algorithm aims only at finding a box $[s_{\text{red}}]$ that is smaller than $[s_{\text{new}}]$ and satisfies (5). It is inspired by (Jaulin and Walter, 1996).

First, a box $[s_{\text{red}}] \subset [s_{\text{new}}]$ is chosen such that $\text{mid}\{[s_{\text{red}}]\} = \text{mid}\{[s_{\text{new}}]\}$ and $\text{rad}\{[s_{\text{red}}]\} = (1-\varepsilon)\cdot\text{rad}\{[s_{\text{new}}]\}$ with $\varepsilon \in]0,1[$, see Fig. 5. Here, $\phi\left([s], u_k, k\Delta t\right)$ represents a box containing the set of all solutions of (1) evaluated at time $(k+1)\,\Delta t$, obtained for an initial state $s \in [s]$ at $k\Delta t$, with a constant control input u_k. Then $[s_{\text{near}}]$ is split into J subboxes $[s_{\text{near}}]_j$, $j = 1, \ldots, J$. For each $[s_{\text{near}}]_j$, one tries to find a constant input $u^j \in [u]$ that robustly drives all states from $[s_{\text{near}}]_j$ to $[s_{\text{red}}]$ (see Fig. 5). For that purpose, one starts from $[u]$. If $\text{mid}\{[u]\}$ robustly drives $[s_{\text{near}}]_j$ to $[s_{\text{red}}]$, then $u^j = \text{mid}\{[u]\}$. Otherwise, $[u]$ is bisected and the midpoints of the two resulting boxes are tested again. The bisection procedure is repeated until a control input is found or until the resulting subboxes are too small to be further bisected. Algorithm 8 summarizes these operations. It has to be called for each subbox $[s_{\text{near}}]_j$ of $[s_{\text{near}}]$.

When control inputs satisfying (5) have been found for each $[s_{\text{near}}]_j$, one may try to reduce $[s_{\text{red}}]$ further.

5. Application in robotics

The proposed Box-RRT algorithm is now applied to path planning for nonholonomic vehicles in a structured 2D environment, where obstacles are described by polygons. One of the difficulties of path planning in this context is the characterization of $\mathbb{S}_{\text{free}}$, which may be quite complex. In (Jaulin, 2001), $\mathbb{S}_{\text{free}}$ is characterized first or constructed

Algorithm 8 find_input(**IN**: $[s]$, $[s_{\text{red}}]$, $[u]$, ε)

$\mathbb{A} \leftarrow [u]$
while $\mathbb{A} \neq \emptyset$ **do**
  $[c] \leftarrow \text{Pop}(\mathbb{A})$
  $[s_{\text{new}}] \leftarrow \phi\left([s], \text{mid}([c]), k \cdot \Delta t\right)$
  if $[s_{\text{new}}] \subseteq [s_{\text{reduced}}]$ **and** collision_free_path($[s_{\text{near}}]$, $[s_{\text{new}}]$, mid($[c]$),Δt) **then**
    return SUCCESS
  else
    if $w([c]_k) < \varepsilon$ **then**
      return FAILURE
    else
      $\{[c_{\text{left}}], [c_{\text{right}}]\} \leftarrow \text{bissection}([c])$
      $\mathbb{A} += [c_{\text{left}}]$
      $\mathbb{A} += [c_{\text{right}}]$
    end if
  end if
end while

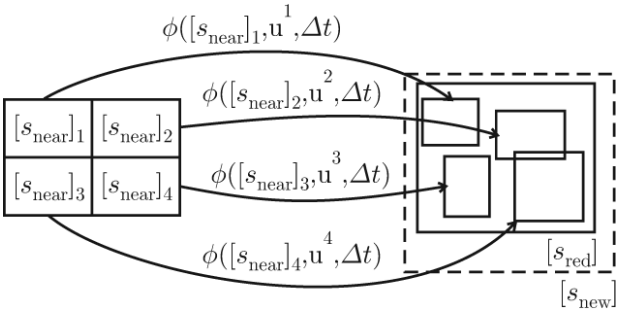


Fig. 5. $[s_{\text{near}}]$ is split into subboxes and for each subbox $[s_{\text{near}}]_j$, an input u^j is computed such that $\phi([s_{\text{near}}]_j, u^j, k\Delta t) \subseteq [s_{\text{red}}]$.

iteratively. Here, $\mathbb{S}_{\text{free}}$ is not explicitely determined: only the constraints of the environment are used to determine whether a set of paths is reliable. Apart from the model of the vehicle considered here, this section provides a description of a collision test to determine whether a set of paths between two consecutive sets of states is reliable.

5.1. Model of the vehicle. Various kinematic or dynamic models of vehicles (Pepy *et al.*, 2006) could be used to test the Set-RRT path planner. Here, a kinematic model based on the classical *simple car* model (LaValle, 2006) evolving in a 2D environment is considered, see Fig. 6. This model incorporates nonholonomic constraints and is given by

$$\begin{cases} \dot{x} = v(1+w_v)\cos\theta, \\ \dot{y} = v(1+w_v)\sin\theta, \\ \dot{\theta} = \frac{v(1+w_v)}{L}\tan\left(\delta(1+w_\delta)\right), \end{cases} \quad (6)$$

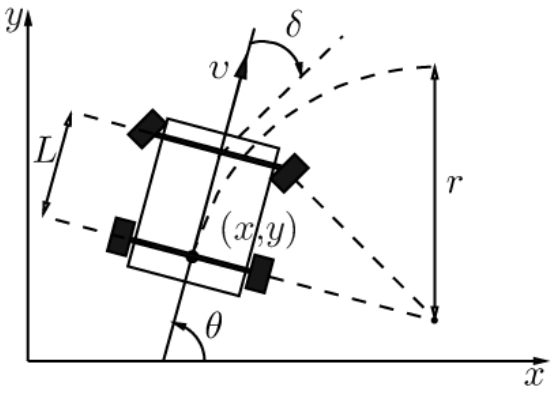


Fig. 6. *Simple car* model.

where the state vector $s = (x, y, \theta)^{\mathrm{T}}$ specifies the position (x, y) and the orientation θ of a frame $\mathcal{V}$ attached to the vehicle with respect to a world frame $\mathcal{W}$ attached to the environment. The control input vector is $u = (v, \delta)^{\mathrm{T}}$, where v denotes the longitudinal speed and $\delta \in [-\delta_{\max}, \delta_{\max}]$ the steering angle. Here, u is assumed to belong to a set $\mathbb{U}$ with a finite number of elements. L is the distance between the front and rear wheels. The noise components $w_v \in [-v_{\mathrm{err}}, v_{\mathrm{err}}]$ and $w_\delta \in [-\delta_{\mathrm{err}}, \delta_{\mathrm{err}}]$ account for the slipping of the vehicle and for the steering imprecision.

In the following figures, walls and obstacles to be avoided are represented by polygons.

5.2. Collision test. If $[s_{\mathrm{init}}]$ and $[s_{\mathrm{goal}}]$ are, respectively, the set of initial and final states, one has to show before starting the path planner that both sets of states belong to $\mathbb{S}_{\mathrm{free}}$. In what follows, the collision tests for a box in the configuration space and for a set of paths between configuration boxes are described. These tests form the core of the `collision_free_path` function used in the box-variant of Algorithms 3 and 4, see Algorithm 9.

5.2.1. Collision-free configuration. The projection of the shape of the vehicle onto the (x, y)-plane in $\mathcal{V}$ is wrapped in a convex polytope $\mathcal{C}$. Each vertex v_i, $i = 1, \ldots, n_{\mathrm{v}}$ of $\mathcal{C}$ is identified by its coordinates $(x_i^{\mathcal{V}}, y_i^{\mathcal{V}})$ in $\mathcal{V}'$, the projection onto the (x, y)-plane of $\mathcal{V}$. Assume that the state of the vehicle is $s = (x, y, \theta)^{\mathrm{T}}$ in $\mathcal{W}$. The boxes $\left([x_i^{\mathcal{W}}], [y_i^{\mathcal{W}}]\right)$ containing the set of coordinates of the n_{v} vertices of the polytope in $\mathcal{W}'$, the projection onto the (x, y)-plane of $\mathcal{W}$, are then

$$\begin{pmatrix} [x_i^{\mathcal{W}}] \\ [y_i^{\mathcal{W}}] \end{pmatrix} = \begin{pmatrix} [x] \\ [y] \end{pmatrix} + \begin{pmatrix} \cos[\theta] & -\sin[\theta] \\ \sin[\theta] & \cos[\theta] \end{pmatrix} \begin{pmatrix} x_i^{\mathcal{V}} \\ y_i^{\mathcal{V}} \end{pmatrix}. \quad (7)$$

To determine the set containing all possible $\mathcal{C}$ in $\mathcal{W}$, one may build the convex envelope of $\left([x_i^{\mathcal{W}}], [y_i^{\mathcal{W}}]\right)$, $i = 1, \ldots, n_{\mathrm{v}}$. A polytope containing these n_{v} boxes is easily obtained by the Graham scan method (Graham, 1972) with time complexity $\mathrm{O}(n \log n)$.

Algorithm 9 collision_free_path($[s_{\mathrm{near}}]$, $[s_{\mathrm{new}}]$, u, Δt, Environment)

```
[s̃1] = [s_near] ⊔ [s_new]
while [s_near] + [0, Δt] f([s̃1], u) ⊄ [s̃1] do
    [s̃1] ← [s̃1] + ε [−1, 1]^{×3}
end while
if CollisionFreeConfiguration([s̃1], u, Δt, Environment) then
    return true;
else
    return false
end if
```

Since this convex hull is an outer approximation of the union of all the possible locations of parts of the vehicle that are associated with a given configuration box, one may now test whether the vehicle is safely located. A collision may occur only if there exists a segment of the polygon that intersects a segment of the environment or if a segment of the environment is entirely included in the polygon.

5.2.2. Collision-free path. The previous test is useful for determining whether $[s_{\mathrm{init}}]$ and $[s_{\mathrm{goal}}]$ are reliable. Now, one has to extend it to determine whether a collision may occur when the vehicle moves from $[s_{\mathrm{near}}]$ to $[s_{\mathrm{new}}]$. This is the aim of the `collision_free_path` function.

Guaranteed numerical integration (Moore, 1966; Lohner, 1987) has been used to obtain $[s_{\mathrm{new}}]$ from $[s_{\mathrm{near}}]$. To enclose the set of trajectories between $[s_{\mathrm{new}}]$ and $[s_{\mathrm{near}}]$, it suffices to find $[\tilde{s}_1]$ satisfying

$$[s_{\mathrm{near}}] + [0, \Delta t] f([\tilde{s}_1], u) \subset [\tilde{s}_1]. \quad (8)$$

Then, the following holds true (Moore, 1966):

$$\forall s \in [s_0] \ \forall t \in [k\Delta t, (k+1)\,\Delta t], \ s(t) \in [\tilde{s}_1]. \quad (9)$$

The box $[\tilde{s}_1]$ is evaluated in the first step of guaranteed numerical integrators (Picard-Lindelöf iteration to prove the existence and uniqueness of solutions to ODEs). It is thus obtained as a byproduct of these integrators. Once $[\tilde{s}_1]$ is computed, it has to be tested for reliability with the same algorithm as for $[s_{\mathrm{init}}]$ and $[s_{\mathrm{goal}}]$. The collision test used with the Box-RRT algorithm is summed up in Algorithm 9.

When it is proved that no collision occurs between any two consecutive nodes of the tree, one proves by induction that the path between $[s_{\mathrm{init}}]$ and $[s_{\mathrm{goal}}]$ is robustly reliable, if it exists.

6. Results

This section provides some results obtained with the Box-RRT algorithm considering the *simple car* model of Section 5.1. In all examples, $\Delta t = 100$ ms. Only projections

of boxes onto the (x, y)-plane are represented to increase readability. All computing times are for a 1.4 GHz Pentium computer.

6.1. Successes. First, results obtained at low speed (lower than $1\ \text{m}\cdot\text{s}^{-1}$) are presented; slipping is then negligible ($v_{\text{err}} = 0$), and it is also assumed that $\delta_{\text{err}} = 0$.

Figure 7(a) represents the solution of a simple path-planning problem using the Box-RRT algorithm. The width of each component of $[s_{\text{init}}]$ is 20 cm for the x and y components and 0.1 rad for θ. The box $[s_{\text{goal}}]$, with size $10\ \text{m}\times 10\ \text{m}\times 2\pi$ rad, has to be reached. The distance between the projections onto the (x, y)-plane of $\text{mid}\{[s_{\text{init}}]\}$ and $\text{mid}\{[s_{\text{goal}}]\}$ is around 100 m. About 30,000 nodes are built in about 28 s by Box-RRT to reach $[s_{\text{goal}}]$. In Fig. 7(b) about 100 000 nodes, built in about 115 s, are required to reach a smaller goal, with size $5\ \text{m}\times 5\ \text{m}\times 2\pi$ rad.

Other types of models could readily be used. For example, Fig. 7(c) (65 000 nodes in about 60 s) shows a path planned for a model only able to turn right. Dynamic vehicle models (Pepy *et al*., 2006) or kinematic chains (Yakey *et al*., 2001) could also be considered.

Harder problems may be solved, such as path planning in an environment with more obstacles, as depicted in Fig. 7(d) (85 000 nodes in about 90 seconds). In this example, the size of $[s_{\text{goal}}]$ is $15\text{m}\times 15\text{m}\times 2\pi$ rad. Again, a guaranteed path between the beginning and the end of the labyrinth is found.

6.2. Challenges. In the previous section, between Fig. 7(a) and 7(b), the size of $[s_{\text{goal}}]$ has been reduced, which made the problem harder to solve. If the size of $[s_{\text{goal}}]$ is reduced further, a path could no longer be found (see Fig. 8(a)) even if it may still exist. Since only prediction is used, and considering the form of the dynamical equation describing the motion of simple car, the size of the box describing the uncertain state always grows along the path. Thus, as soon as the size of $[s]$ at the end of a path becomes bigger than that of $[s_{\text{goal}}]$, it is impossible to reach $[s_{\text{goal}}]$ from this box.

The same problem appears when the skidding error is too large. This problem is illustrated in Fig. 8(b), where the size of $[s_{\text{init}}]$ is $10\ \text{cm}\times 10\ \text{cm}\times [1, 1.05]$ rad, the size of $[s_{\text{goal}}]$ is $10\ \text{m}\times 10\ \text{m}\times 2\pi$ rad, $v_{\text{err}} = 10^{-2}$, and $\delta_{\text{err}} = 10^{-3}$. Uncertainty then becomes exceedingly large and the vehicle can no longer be guaranteed to pass through the corridor. Thus, this problem cannot be solved using the version of Box-RRT presented in Section 4.2, unless some exteroceptive measurements are used at some points along the path to reduce uncertainty.

6.3. Application of Reach-RRT. The same simulated conditions are considered as in Fig. 8(b) of Section 6.2.

Results illustrated in Figs. 9(a) and 9(b) show that the use of differentiated inputs allows a reduction of the boxes and a proof of guaranteed reachability.

In this example, adapting the input allows the size of the box $[s_{\text{new}}]$ at each iteration to be reduced by 17% on average. This rate is obtained at the price of splitting each $[s_{\text{near}}]$ in at least 64 subboxes, which significantly increases the computational load. Thus, the reduction step may be used with a period larger than Δt.

In Fig. 9(a), box reduction is performed every second. The path planner finds a path by generating about 10 000 nodes. Similarly, a path is found for the problem illustrated in Fig. 9(b) with box reduction performed every two seconds on each path.

As mentioned earlier, with Reach-RRT, one proves that for each initial state a control input exists that is able to drive the system robustly to the goal area. It is then worth trying to divide the global path planning task into several local (short-term) planning tasks along the path obtained by Reach-RRT. This allows information provided by sensors to be taken into account, facilitating the task of Box-RRT.

7. Conclusions and perspectives

This paper has presented algorithms based on rapidly-exploring random trees able to perform path planning tasks for models of systems including uncertainties. Uncertain quantities are assumed to belong to sets. A first conceptual Set-RRT path planner dealing with general sets has been presented, followed by an implementable Box-RRT dealing with boxes. Box-RRT has also been adapted to perform reachability analysis.

Some algorithms presented in this paper are rather preliminary, but show the potential of the approach. For example, the choice of the control input in Box-RRT or Reach-RRT is not optimised yet. Better local, i.e., short-term, reachability analysis techniques could be used, see, e.g., (Collins and Goldsztejn, 2008; Ramdani *et al*., 2008).

In the present version of Set-RRT and Reach-RRT, $\mathbb{S}_{\text{free}}$ is assumed to be constant with time. One could easily adapt the proposed algorithms to $\mathbb{S}_{\text{free}}$ varying with time, to describe moving obstacles, to take into account the limited energy available to the system, etc.

The model of the mobile robot used in this paper is extremely simple, and does not take info account the vehicle dynamics. As a result, the paths generated may exhibit abrupt changes. A natural way of improving the smoothness of the trajectories generated would be to model the vehicle dynamics and put constraints on acceleration. This should form the subject of future studies.

References

Ackermann, J., Barlett, A., Kaesbauer, D., Sienel, W. and Steinhauser, R. (1993). *Robust Control Systems with Uncertain*

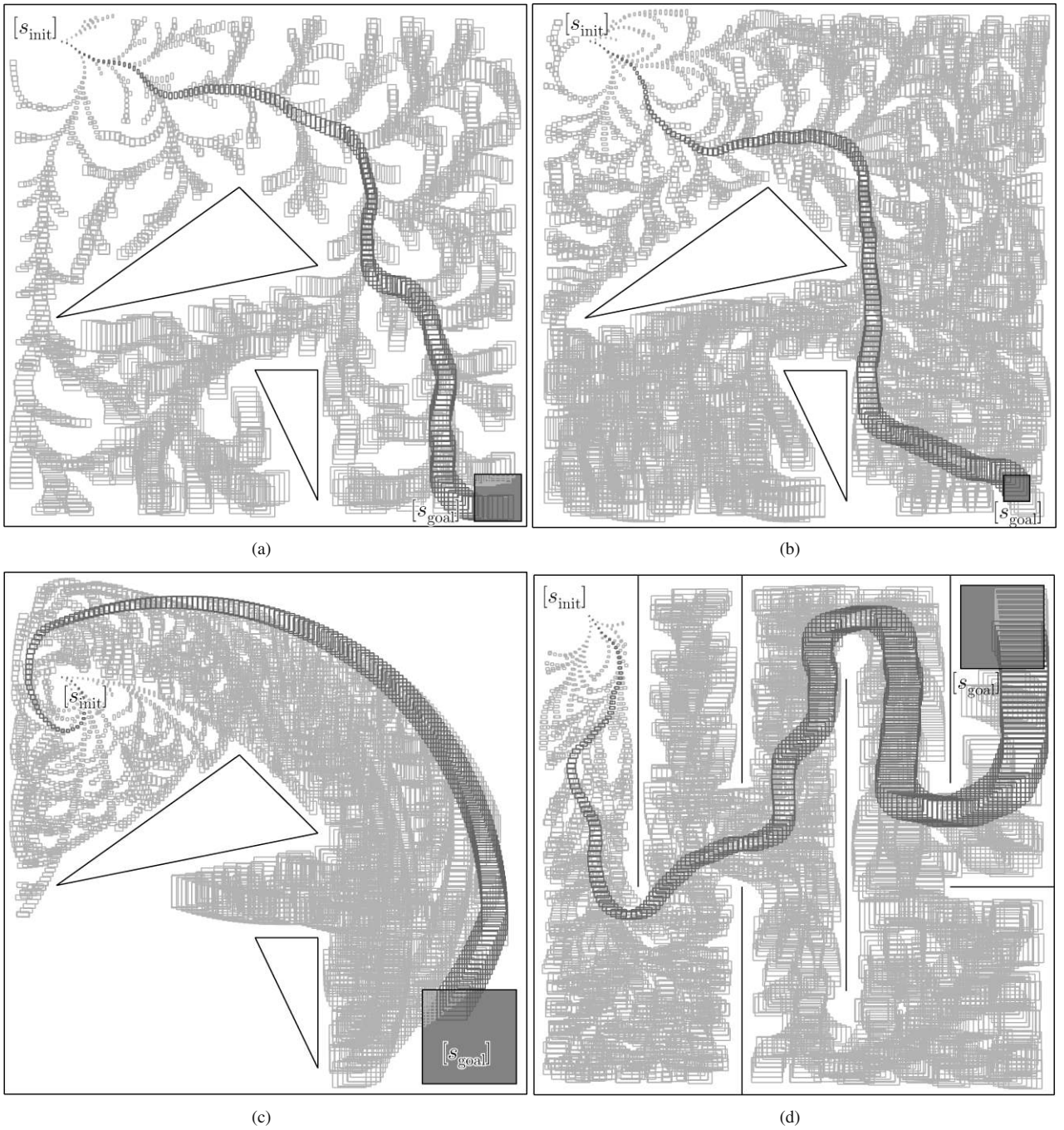


Fig. 7. Some path-planning problems solved by Box-RRT: (a) few obstacles and a large goal area, (b) few obstacles and a small goal area, (c) car which can only turn right, (d) labyrinthic environment.

Physical Parameters, Springer-Verlag, London.

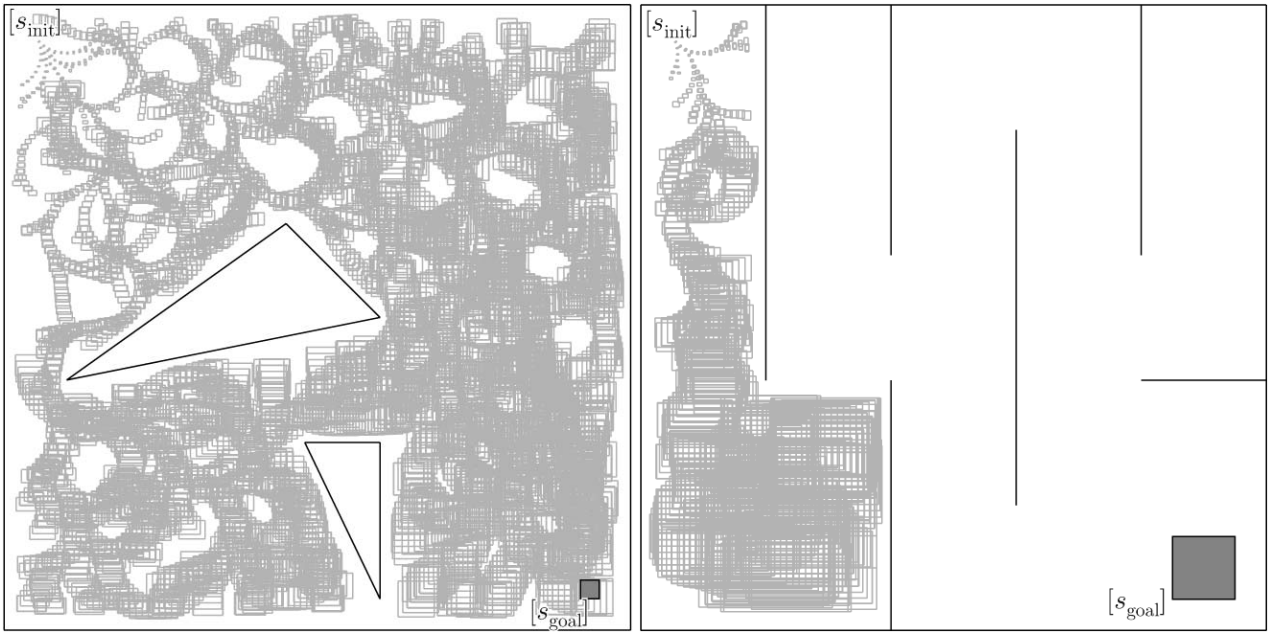
Alamo, T., Bravo, J., Camacho, E. and de Sevilla, U. (2003). Guaranteed state estimation by zonotopes, *Proceedings of the 42nd Conference on Decision and Control, Maui, Hi,* pp. 1035–1043.

Berger, M. (1987). *Geometry I and II*, Springer-Verlag, Berlin.

Bouilly, B., Simeon, T. and Alami, R. (1995). A numerical technique for planning motion strategies of a mobile robot in presence of uncertainty, *Proceedings of the IEEE International Conference on Robotics and Automation, Nagoya, Japan,* pp. 1327–1332.

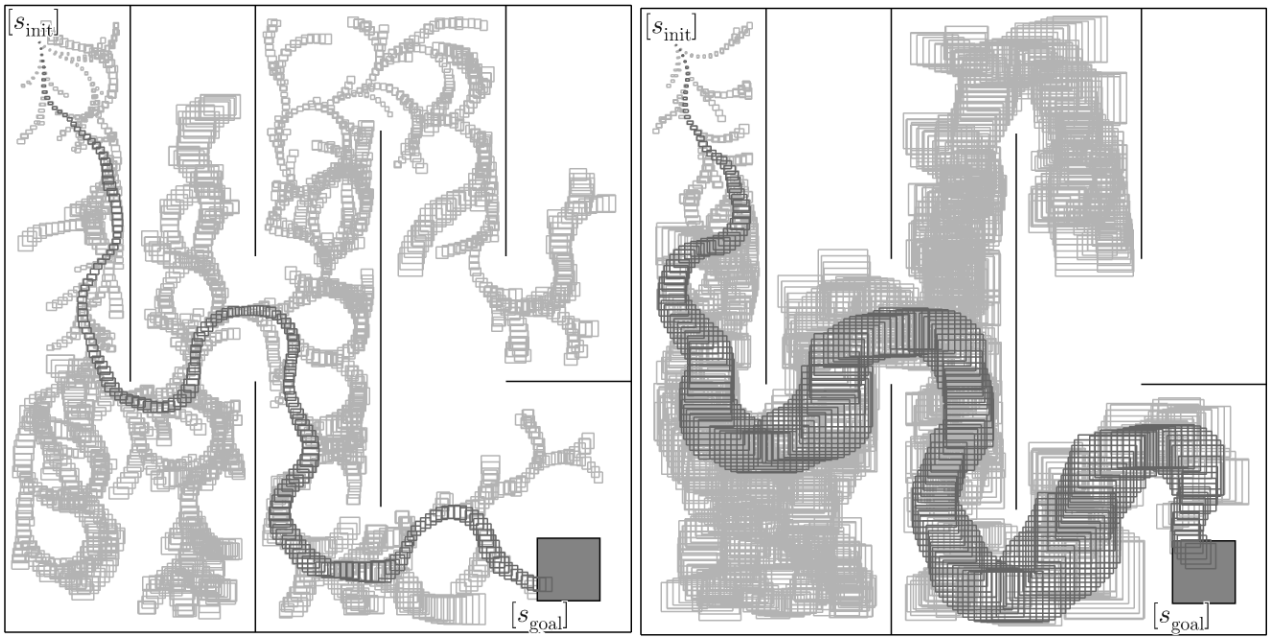
Collins, P. and Goldsztejn, A. (2008). The reach-and-evolve algorithm for reachability analysis of nonlinear dynamical systems, *Electronic Notes in Theoretical Computer Science* **223**: 87–102.

Fraichard, T. and Mermond, R. (1998). Path planning with uncertainty for car-like robots, *Proceedings of the IEEE Inter-*



(a) (b)

Fig. 8. Some more difficult problems where Box-RRT fails to find a solution: (a) Box-RRT fails to find a path (the goal area is too small), (b) Box-RRT fails to find a path (skidding errors are too large).



(a) (b)

Fig. 9. Results obtained using the FindInput algorithm: (a) path found using box reduction every second, (b) path found using box reduction every two seconds.

national Conference on Robotics and Automation, Leuven, Belgium, pp. 27–32.

Francis, B. A. and Khargonekar, P. P. (Eds.) (1995). *Robust Control Theory*, IMA Volumes in Mathematics and Its Applications, Vol. 66, Springer-Verlag, New York, NY.

Gonzalez, J. P. and Stentz, A. (2004). Planning with uncertainty in position: An optimal planner, *Technical Report CMU-RI-TR-04-63*, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Gonzalez, J. P. and Stentz, A. (2005). Planning with uncertainty in position: An optimal and efficient planner, *Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Edmonton, Canada,* pp. 2435–2442.

Gonzalez, J. P. and Stentz, A. (2007). Planning with uncertainty in position using high-resolution maps, *Proceedings of the IEEE International Conference on Robotics and Automation, Rome, Italy,* pp. 1015–1022.

Graham, R. L. (1972). An efficient algorithm for determining the convex hull of a finite planar set, *Information Processing Letters* **1**(4): 132–133.

Jaulin, L. (2001). Path planning using intervals and graphs, *Reliable Computing* **7**(1): 1–15.

Jaulin, L. (2002). Nonlinear bounded-error state estimation of continuous-time systems, *Automatica* **38**(6): 1079–1082.

Jaulin, L., Kieffer, M., Didrit, O. and Walter, E. (2001). *Applied Interval Analysis*, Springer-Verlag, London.

Jaulin, L. and Walter, E. (1996). Guaranteed tuning, with application to robust control and motion planning, *Automatica* **32**(8): 1217–1221.

Kieffer, M., Jaulin, L., Braems, I. and Walter, E. (2001). Guaranteed set computation with subpavings, *in* W. Kraemer and J. W. von Gudenberg (Eds.), *Scientific Computing, Validated Numerics, Interval Methods*, Kluwer Academic/Plenum Publishers, New York, NY, pp. 167–178.

Kieffer, M., Jaulin, L. and Walter, E. (2002). Guaranteed recursive nonlinear state bounding using interval analysis, *International Journal of Adaptative Control and Signal Processing* **6**(3): 193–218.

Kieffer, M. and Walter, E. (2003). Nonlinear parameter and state estimation for cooperative systems in a bounded-error context, *in* R. Alt, A. Frommer, R. B. Kearfott and W. Luther (Eds.), *Numerical Software with Result Verification (Platforms, Algorithms, Applications in Engineering, Physics, and Economics)*, Springer, New York, NY, pp. 107–123.

Kieffer, M. and Walter, E. (2006). Guaranteed nonlinear state estimation for continuous-time dynamical models from discrete-time measurements, *Proceedings of the 6th IFAC Symposium on Robust Control, Toulouse, France,* (on CD-ROM).

Kuffner, J. J. and LaValle, S. M. (2000). RRT-connect: An efficient approach to single-query path planning, *Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, USA,* pp. 995–1001.

Lambert, A. and Gruyer, D. (2003). Safe path planning in an uncertain-configuration space, *Proceedings of the IEEE International Conference on Robotics and Automation, Taipei, Taiwan,* pp. 4185–4190.

Latombe, J. C. (1991). *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA.

LaValle, S. M. (1998). Rapidly-exploring Random Trees: A new tool for path planning, *Technical report*, Iowa State University, Ames, IO.

LaValle, S. M. (2006). *Planning Algorithms*, Cambridge University Press, Cambridge, Available at: `http://planning.cs.uiuc.edu/`.

LaValle, S. M. and Kuffner, J. J. (2001a). Randomized kinodynamic planning, *International Journal of Robotics Research* **20**(5): 378–400.

LaValle, S. M. and Kuffner, J. J. (2001b). Rapidly-exploring random trees: Progress and Prospects, *in* B. R. Donald, K. M. Lynch and D. Rus (Eds.), *Algorithmic and Computational Robotics: New Directions*, A. K. Peters, Wellesley, MA, pp. 293–308.

Lazanas, A. and Latombe, J. C. (1995). Motion planning with uncertainty: A landmark approach, *Artificial Intelligence* **76**(1-2): 287–317.

Lohner, R. (1987). Enclosing the solutions of ordinary initial and boundary value problems, *in* E. Kaucher, U. Kulisch and C. Ullrich (Eds.), *Computer Arithmetic: Scientific Computation and Programming Languages*, BG Teubner, Stuttgart, pp. 255–286.

Luenberger, D. (1966). Observers for multivariable systems, *IEEE Transactions on Automatic Control* **11**(2): 190–197.

Moore, R. E. (1966). *Interval Analysis*, Prentice-Hall, Englewood Cliffs, NJ.

Moore, R. E. (1979). *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, PA.

Pepy, R. and Lambert, A. (2006). Safe path planning in an uncertain-configuration space using RRT, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China,* pp. 5376–5381.

Pepy, R., Lambert, A. and Mounier, H. (2006). Reducing navigation errors by planning with realistic vehicle model, *Proceedings of the IEEE Intelligent Vehicle Symposium, Tokyo, Japan,* pp. 300–307.

Raissi, T., Ramdani, N. and Candau, Y. (2004). Set membership state and parameter estimation for systems described by nonlinear differential equations, *Automatica* **40**(10): 1771–1777.

Ramdani, N., Meslem, N. and Candau, Y. (2008). Reachability analysis of uncertain nonlinear systems using guaranteed set integration, *Proceedings of the IFAC World Congress, Seoul, Korea.*

Schweppe, F. C. (1973). *Uncertain Dynamic Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Yakey, J., LaValle, S. M. and Kavraki, L. E. (2001). Randomized path planning for linkages with closed kinematic chains, *IEEE Transactions on Robotics and Automation* **17**(6): 951–958.

Romain Pepy was awarded an engineering degree in computer science from Ecole Supérieure d'Ingénieurs en Electrotechnique et Electronique of Amiens, France, in 2005 and a Ph.D. degree in control and signal processing from Paris-Sud University in 2009. His current research interests include state estimation and robust motion planning with application to aerial robotics. He is a research scientist at ONERA, a French public research facility dedicated to aerospace.

Michel Kieffer received *Agrégation* in applied physics at Ecole Normale Supérieure de Cachan, France. He received a Ph.D. in control and signal processing in 1999, and *Habilitation à Diriger des Recherches* in 2005, both from Paris-Sud University. He is an assistant professor in signal processing for communications at Paris-Sud University and a researcher at Laboratoire des Signaux et Systèmes. His research interests are in joint source-channel coding and decoding techniques for reliable transmission of multimedia contents. He is also interested in guaranteed parameter and state estimation for systems described by non-linear models. He has co-authored more than 80 contributions in journals, conference proceedings, or books. He is one of the co-authors of the book *Applied Interval Analysis* published by Springer-Verlag in 2001.

Eric Walter, a physicist by training, was awarded *Doctorat d'État* in control theory in 1980 by Paris-Sud University. He is the director for research at CNRS (the French national center for scientific research). His research interests revolve around parameter estimation and its application to chemical engineering, chemistry, control, image processing, medicine pharmacokinetics, and robotics. He is the author or co-author of more than 200 papers in refereed journals and conferences, and of three books, namely, *Identifiability of State-Space Models* (Springer, Berlin, 1982), *Identification of Parametric Models from Experimental Data* (Springer, London, 1997, with Luc Pronzato), and *Applied Interval Analysis* (Springer, London, 2001, with Luc Jaulin, Michel Kieffer, and Olivier Didrit). He is presently the director of Laboratoire des Signaux et Systèmes, a research facility hosting more than 100 persons and common to CNRS, the school of electrical engineering Supélec and Paris-Sud University.

Received: 16 September 2008
Revised: 9 February 2009