Extended Reliable Robust Motion Planners

Adina M Panchea, Alexandre Chapoutot, David Filliat

Extended Reliable Robust Motion Planners *

Adina M. Panchea †, Alexandre Chapoutot and David Filliat‡

March 21, 2017

Abstract

A new method to plan guaranteed to be safe paths in an uncertain environment, with an uncertain initial and final configuration space, while avoiding static obstacles is presented. First, two improved versions of the previously proposed BoxRRT algorithm are presented: both with a better integration scheme and one of them with the control input selected according to a desired objective, and not randomly, as in the original formulation. Second, a new motion planner, called towards BoxRRT*, based on optimal Rapidly-exploring Random Trees algorithm and using interval analysis is introduced. Finally, each of the described algorithms are evaluated on a numerical example. Results show that our algorithms make it possible to find shorter reliable paths with less iterations.

1 Introduction

The motion planning problem, as addressed in this paper, consists in finding a path, or a sequence of control policy which drives a mobile robot, with a given dynamics description, from a given initial state region to a given goal region while avoiding collisions with a given set of obstacles.

The development of efficient and intelligent motion planning algorithms used for autonomous navigation, which is at the core of autonomous mobile robotic devices, remains a challenge in particular when trying to guarantee the safety of the vehicle. The guarantee of vehicle's safety implies that the motion planning should take into account uncertainties usually resulting from the approximate initial mobile robot localization, from imperfect embedded sensors or from the approximate models used to describe the behaviour of the robotic devices.

Related work

Many motion planning algorithms have been proposed in the literature. When dealing with complex environments, non-holonomic vehicles or high-dimensional state-space, a popular approach is to use stochastic sampling to discretize the configuration space. Among the existing approaches, we focus on the Rapidly-exploring Random Trees

*This work was supported by DGA MRIS.

†Adina M. Panchea is with COSYNUS/LIX UMR 7161, Ecole Polytechnique, Palaiseau, France panchea@lix.polytechnique.fr

‡Alexandre Chapoutot and David Filliat are with the U2IS, ENSTA ParisTech, Univ. Paris-Saclay, Palaiseau, France {chapoutot, filliat}@ensta-paristech.fr

(RRT) ([12, 14, 10, 13]) path planning algorithm and its many variants. These algorithms have the advantage of rapidly covering the whole configuration space and of easily integrating complex robot models.

However, RRT lacks in guaranteeing an optimal solution. The (asymptotic) optimality of the solution is provided by the optimal Rapidly-exploring Random Trees (RRT*) first proposed in [8] and recently used, for example to plan trajectories for aerial vehicle ([24, 23]). In [18], a survey on motion planning algorithms based on RRT* is given.

Uncertainty in path planning has been considered using several representations such as set-membership ([19, 21]) or covariance matrices ([11, 22, 4]). While the latter is able to find paths with a collision probability under a given threshold, set-membership approaches can guarantee safe trajectories under a bounded noise assumption.

The localization information provided by imperfect proprioceptive sensors, while represented by Gaussian functions ([11, 22]) can guarantee the safety of the path at a certain confidence threshold. Recently, [20] provided the guarantee of a safe path to imperfect proprioceptive sensors, while considering the uncertainties bounded with know bounds. Under the latter uncertainty representation, [21] introduced a preliminary conceptual reliable and robust path planner based on RRT principles and solved in an interval analysis ([15, 7]) framework. The interval analysis framework was previously used ([6]) along with graph algorithms to find collision-free short paths in a given configuration space.

Contributions

This study presents motion planning algorithms which can guarantee safe paths in an uncertain configuration space, where all approximate initial and final mobile robot localisation are bounded with known bounds. Improving the motion planner (denoted BoxRRT) proposed in [21], our **first contribution** consists in an improved BoxRRT planner which makes use of modern and new tools ([1, 2]) for the guaranteed numerical integration employed by the motion planner combined with methods coming from constraint satisfaction problems. The **second contribution** consists in a second improved BoxRRT algorithm which makes use of the guaranteed numerical integration improvements along with the choice of the control input according to a desired objective and not randomly, as already proposed in the literature. As a **third contribution**, a preliminary attempt towards a new reliable and robust motion planning algorithm based on RRT* principles is introduced.

This paper is organized as follows. First, Sect. 2 introduces the problem formulation, while Sect. 3 provides two improved versions of BoxRRT algorithm. Next, Sect. 4 describes the new planner based on RRT*. The resulting three proposed reliable and robust path planners are applied to plan paths of a non-holonomic vehicle in Sect. 5, where the simulations results are provided. Finally, some concluding remarks and perspectives are drawn in Section 6.

2 Problem Statement

This paper considers a mobile robot which has to be driven in a two-dimensional static environment from an initial state to a desired one while avoiding obstacles represented by polygons shapes.

The configuration space $\mathbb{S} = \mathbb{S}_{\text{free}} \cup \mathbb{S}_{\text{obs}}$ is therefore composed of two subsets: the free region subset $\mathbb{S}_{\text{free}} = \mathbb{S} \setminus \mathbb{S}_{\text{obs}}$ where the mobile robot is allowed to move and the obstacle region subset $\mathbb{S}_{\text{obs}}$ which the mobile robot needs to avoid. Moreover, uncertainties related to its initial and final position and orientation w.r.t. a frame attached to the environment are considered.

2.1 Problem formulation

Consider the differential system which can describe the evolution of a mobile robot system:

$$\dot{\mathbf{s}}(t) = \mathbf{f}(\mathbf{s}(t), \mathbf{u}(t)) \tag{1}$$

where $\mathbf{s} \in \mathbb{S}$ is considered to be the measurable state of the system, while $\mathbf{u}(t) \in \mathbb{U}$ is the admissible control input. The exact solution of (1) from the inital condition $\mathbf{s}_0$ is denoted by $\mathbf{s}(t;\mathbf{s}_0)$. From an initial state $\mathbf{s}_0$ which belongs to a known set $\mathbf{s}_0 \in \mathbb{S}_{\text{init}} \subset \mathbb{S}_{\text{free}}$ the system needs to reach a given set of goal states $\mathbb{S}_{\text{goal}} \subset \mathbb{S}_{\text{free}}$. The problem formulation comes from [21].

The purpose of the *robust motion planner* is to provide a sequence of control inputs $\mathbf{u} \in \mathbb{U}_{[\mathbf{u}]}^{\Delta t}$ bounded over intervals of time $[K\Delta t, (K+1)\Delta t[$, with $\Delta t > 0$ and $K \in \mathbb{N}$, which will drive the system to reach $\mathbb{S}_{\text{goal}}$ while avoiding the non-admissible states $\mathbb{S}_{\text{obs}}$ whatever the initial state $\mathbf{s} \in \mathbb{S}_{\text{init}}$ are. If such a sequence of control input $\mathbf{u} \in \mathbb{U}_{[\mathbf{u}]}^{\Delta t}$ is proved to drive the system from any initial state $\mathbf{s} \in \mathbb{S}_{\text{init}}$ to a final state in $\mathbb{S}_{\text{goal}}$ then the found robust planned path is *reliable*.

The formulation of such a robust motion planner for which there exists a sequence of control input $\mathbf{u} \in \mathbb{U}_{[\mathbf{u}]}^{\Delta t}$ to drive the system from an uncertain initial state to a set of goal states $\mathbb{S}_{\text{goal}}$ is as follows:

$$\begin{array}{l} \exists K > 0 \text{ and } \mathbf{u} \in \mathbb{U} \text{ such that} \\ \forall \mathbf{s}_0 \in \mathbb{S}_{\text{init}},\ \forall\, \mathbf{s}(K\Delta t;\mathbf{s}_0) \in \mathbb{S}_{\text{goal}} \text{ and} \\ \forall t \in [0, K\Delta t],\ \mathbf{s}(t;\mathbf{s}_0) \in \mathbb{S}_{\text{free}}, \end{array} \tag{2}$$

with $\mathbf{s}(t)$ the solution of (1).

3 Motion Planner Algorithms

This section recalls all necessary notation deployed in this study regarding *interval vectors* or *boxes* ([7]) which are being used to represent the environment uncertainties.

Next, two new versions of BoxRRT motion planner: **the random control input BoxRRT (*rciBoxRRT*) motion planner algorithm** and **the selected control input BoxRRT (*sciBoxRRT*) motion planner algorithm** are introduced. Both algorithms are based on RRT motion planner ([12, 13, 14, 10]) which is an incremental-based method with the purpose of efficiently explore all the given configuration space from a given starting configuration.

The idea of BoxRRT is not new, being previously proposed by [21, 20] for the case where the uncertainties related to the configuration space are considered only on the final state and not on the initial configuration state. While in this study the interest is to consider uncertainties in the initial and final configuration state space. Moreover, the improvements made on our new versions of BoxRRT are presented in the followings.

3.1 Interval analysis

A scalar (real) interval $[x] = [\underline{x}, \overline{x}]$ is a closed and connected subset of $\mathbb{R}$, where $\underline{x}$ represents the lower bound and $\overline{x}$ represents the upper bound. Two intervals $[u]$ and $[v]$ are equal if and only if $\underline{u} = \underline{v}$ and $\overline{u} = \overline{v}$. An interval vector (or box) $[\mathbf{x}]$ is a subset of $\mathbb{R}^n$ which is the Cartesian product of scalar intervals $[\mathbf{x}] = [x_1] \times [x_2] \times \cdots \times [x_n]$, where the ith component is the projection of $[\mathbf{x}]$ onto the ith axis. The interval hull of a set $\mathbb{A}$ is the smallest box which contains $\mathbb{A}$, denoted by Hull($\mathbb{A}$). The inner approximation of a set $\mathbb{A}$, denoted Int($\mathbb{A}$), is a box included in $\mathbb{A}$, *i.e*, Int($\mathbb{A}$) $\subset \mathbb{A}$. The Hausdorff distance [7, 17] of two intervals $[x_1]$ and $[x_2]$ is

$$d([x_1],[x_2]) = \sup\{|\underline{x}_1 - \underline{x}_2|, |\overline{x}_1 - \overline{x}_2|\} \tag{3}$$

Validated numerical integration methods are interval counterpart of numerical integration methods. A validated numerical integration of a differential equation, as defined in (1) assuming piece-wise constant input, consists in a discretization of time, such that $t_0 \leqslant \cdots \leqslant t_{\text{end}}$, and a computation of enclosures of the set of states of the system $\mathbf{s}_0, \ldots, \mathbf{s}_{\text{end}}$, by the help of a guaranteed integration scheme. In details, a guaranteed integration scheme is made of:

- an integration method $\Phi(f, \mathbf{s}_j, t_j, h)$, starting from an initial value $\mathbf{s}_j$ at time t_j and a finite time horizon h (the step-size), producing an approximation $\mathbf{s}_{j+1}$ at time $t_{j+1} = t_j + h$, of the exact solution $\mathbf{s}(t_{j+1}; \mathbf{s}_j)$, *i.e.*, $\mathbf{s}(t_{j+1}; \mathbf{s}_j) \approx \Phi(f, \mathbf{s}_j, t_j, h)$;
- a truncation error function $\text{lte}_\Phi(d, \mathbf{s}_j, t_j, h)$, such that $\mathbf{s}(t_{j+1}; \mathbf{s}_j) = \Phi(f, \mathbf{s}_j, t_j, h) + \text{lte}_\Phi(f, \mathbf{s}_j, t_j, h)$.

Our validated numerical integration method is a two step method starting at time t_j and for which *i*) it computes an enclosure $[\tilde{\mathbf{s}}_j]$ of the solution of (1) over the time interval $[t_j, t_{j+1}]$ to bound $\text{lte}_\Phi(d, \mathbf{s}_j, t_j, h)$; *ii*) it computes a tight enclosure of the solution of (1) for the particular time instant t_{j+1}. There are many methods for these two steps among Taylor series and Runge-Kutta methods see [16, 1] and the references therein for more details.

As a result, validated numerical integration methods produce two functions depending on time

$$R : \left\{ \begin{array}{c} \mathbb{R} \mapsto \mathbb{IR}^n \\ t \to [\mathbf{s}] \end{array} \right. \tag{4}$$

with for a given t_i, $R(t_i) = \{\mathbf{s}(t_i; \mathbf{s}_0) : \forall \mathbf{s}_0 \in [\mathbf{s}_0]\} \subseteq [\mathbf{s}]$, and

$$\tilde{R} : \left\{ \begin{array}{c} \mathbb{IR} \mapsto \mathbb{IR}^n \\ [\underline{t}, \overline{t}] \to [\tilde{\mathbf{s}}] \end{array} \right. \tag{5}$$

with $\tilde{R}([\underline{t}, \overline{t}]) = \{\mathbf{s}(t; \mathbf{s}_0) : \forall \mathbf{s}_0 \in [\mathbf{s}_0] \wedge \forall t \in [\underline{t}, \overline{t}]\} \subseteq [\tilde{\mathbf{s}}]$.

3.2 The *rciBoxRRT* and *sciBoxRRT* proposed motion planners

Let's start by introducing the global description which is the same for both algorithms. Next, the algorithm which gathers improvements regarding to the previously proposed BoxRRT algorithm and which has the same formulation for both our new versions of BoxRRT motion planner is introduced. Finally, each procedure of the algorithm is explained separately. The difference between the proposed algorithms lies in the choice of the control input and it will be explained as follows.

***Description*:** First the given initial configuration $[\mathbf{s}_{\text{init}}]$ is added to the exploration tree G (Line 1). Then, a state $[\mathbf{s}_{\text{rand}}] \subset \mathbb{S}_{\text{free}}$ is randomly chosen by the procedure *random-box-GoalBias* (Line 4). The *nearest-neighbor* procedure from Line 5 returns the closest vertex $[\mathbf{s}_{\text{near}}]$ to $[\mathbf{s}_{\text{rand}}]$ in the tree G, according to a certain metric d. A control input $\mathbf{u} \in [\mathbf{u}]$ is chosen according to a specified criterion or randomly through the *select input* procedure. Then, in the *prediction* procedure, (1) is integrated over a fix time interval Δt with the initial condition $[\mathbf{s}_{\text{near}}]$ and a constant control input $\mathbf{u}$ (given at Line 6) and will result in a new state $[\mathbf{s}_{\text{new}}]$ (Line 7). If it can be proved that all state values along the trajectory between $[\mathbf{s}_{\text{near}}]$ and $[\mathbf{s}_{\text{new}}]$ lie in $\mathbb{S}_{\text{free}}$ being a *collision free path*, then the path between $[\mathbf{s}_{\text{near}}]$ and $[\mathbf{s}_{\text{new}}]$ is considered reliable and $[\mathbf{s}_{\text{new}}]$ is added to G as a new vertex and connected to its parent $[\mathbf{s}_{\text{near}}]$ though the *G.add-guaranteed-vertex* procedure. Otherwise, $[\mathbf{s}_{\text{new}}]$ is not added to G. Lines 4 to 11 are repeated until a chosen number of iterations MaxIter is reached or until a path is found meaning $[\mathbf{s}_{\text{new}}] = [\mathbf{s}_{\text{goal}}]$, or most likely when $[\mathbf{s}_{\text{new}}] \subset [\mathbf{s}_{\text{goal}}]$. Note that we have $[\mathbf{s}_{\text{init}}] = \text{Hull}(\mathbb{S}_{\text{init}})$, $[\mathbf{s}_{\text{obs}}] = \text{Hull}(\mathbb{S}_{\text{obs}})$ and $[\mathbf{s}_{\text{goal}}] = \text{Int}(\mathbb{S}_{\text{goal}})$ to ensure the soundness of the proposed algorithm.

input : $[\mathbf{s}_{\text{init}}] \subset \mathbb{S}_{\text{free}}$, $[\mathbf{s}_{\text{goal}}] \subset \mathbb{S}_{\text{free}}$, $\Delta t \in \mathbb{R}^+$, MaxIter $\in \mathbb{N}$
output: G

1 $G.init([\mathbf{s}_{\text{init}}])$;
2 $i \leftarrow 0$;
3 **repeat**
4 $\quad [\mathbf{s}_{\text{rand}}] \leftarrow$ random-box-*GoalBias*$(\mathbb{S}_{\text{free}})$;
5 $\quad [\mathbf{s}_{\text{near}}] \leftarrow$ nearest-neighbor$(G, [\mathbf{s}_{\text{rand}}])$;
6 $\quad \mathbf{u} \leftarrow$ select input$([\mathbf{s}_{\text{rand}}], [\mathbf{s}_{\text{near}}])$;
7 $\quad [\mathbf{s}_{\text{new}}] \leftarrow$ prediction$([\mathbf{s}_{\text{near}}], \mathbf{u}, \Delta t)$;
8 $\quad$ **if** *collision free path* $([\mathbf{s}_{near}], [\mathbf{s}_{new}], \mathbf{u}, \Delta t)$ **then**
9 $\quad\quad G$.add-guaranteed-vertex$([\mathbf{s}_{\text{near}}], [\mathbf{s}_{\text{new}}], \mathbf{u})$;
10 $\quad\quad$ **return** $[\mathbf{s}_{\text{new}}]$
11 $\quad$ **return** $\emptyset$
12 **until** $i++ <$ *MaxIter* **or** $([\mathbf{s}_{new}] \neq \emptyset$ **and** $[\mathbf{s}_{new}] \subset [\mathbf{s}_{goal}])$;
13 **return** G

Algorithm 1: BoxRRT motion planner algorithm

***Random box GoalBias procedure*:** This procedure, previously proposed in [21], consists in choosing the random state in the final configuration state $[\mathbf{s}_{\text{rand}}] \subset [\mathbf{s}_{\text{goal}}]$ with a probability $p > 0$. Other techniques towards a more improved random procedure can be thought of, such as the use of the artificial potential field algorithms (APFs) as proposed in [24, 23].
***Nearest neighbor procedure*:** Finds the closest vertex to the $[\mathbf{s}_{\text{rand}}]$ one according to a chosen metric d. The Hausdorff distance between two intervals as defined in (3) is considered.
***Prediction procedure*:** Finds a new state $[\mathbf{s}_{\text{new}}]$ while integrating (1) with the selected control input, given by the select input procedure, over an interval of time Δt. This step is based on validated numerical integration methods as explained in Section 3.1 and using function $R(t)$.
***Collision free path procedure*:** If $[\mathbf{s}_{\text{init}}]$ and $[\mathbf{s}_{\text{goal}}]$ are, respectively, the imperfect initial and final states, one has to show before starting the path planner that both sets of states belong to $\mathbb{S}_{\text{free}}$. When it is proved that no collision occurs between any two consecutive vertices of the tree, one proves by induction that the path between $[\mathbf{s}_{\text{init}}]$ and $[\mathbf{s}_{\text{goal}}]$ is

robustly reliable, if it exists. The techniques used in this procedure are based on new tool and functions proposed by [3], which are capable of testing during the integration procedure if a collision occurred. Therefore this procedure differs from the previously BoxRRT one which uses wrap techniques [21, 20]. More precisely, using the enclosure $\tilde{R}(t)$ of the trajectory of (1), checking that no collision occurs is simply an interval test which checks if $\tilde{R}(t)$ does not intersect $[\mathbf{s}_{\text{obs}}]$ for all t.

Select input procedure*:** This procedure is used to find a control input which finds a new state starting from a given initial state. The difference between the two new versions of BoxRRT is made in this procedure: the ***rciBoxRRT motion planner uses a control input chosen randomly among the set of admissible values $\mathbf{u} \in \mathbb{U}$, while ***sciBoxRRT*** motion planner uses a designed control input according to a desired behaviour or to a chosen criterion (see Section 5 for an example).

4 Towards BoxRRT* Motion Planner Algorithm

This section introduces a new reliable robust path planner for uncertain environments based on optimal RRTs (RRT*)([8, 24, 23]), which is denoted ***tBoxRRT****. A general description of the algorithm is presented, followed by the description of the used procedures.

***Description*:** As in RRT, the tree G is initialized with the given initial configuration $[\mathbf{s}_{\text{init}}]$. Then, a state $[\mathbf{s}_{\text{rand}}] \in \mathbb{S}_{\text{free}}$ is randomly chosen by *random-box-GoalBias* procedure and its nearest vertex $[\mathbf{s}_{\text{nearest}}]$ according to a defined metric d is provided by the *nearest-neighbor* procedure. *Steer procedure* designs a control input according to a desired behaviour or according to a specific criterion. Eq. (1) is integrated over a fixed time interval Δt with the initial condition $[\mathbf{s}_{\text{nearest}}]$ and a constant control input $\mathbf{u}$. Then a new state $[\mathbf{s}_{\text{new}}]$ is found. If the trajectory between $[\mathbf{s}_{\text{nearest}}]$ and $[\mathbf{s}_{\text{new}}]$ lie in $\mathbb{S}_{\text{free}}$, then the path between $[\mathbf{s}_{\text{nearest}}]$ and $[\mathbf{s}_{\text{new}}]$ is reliable and $[\mathbf{s}_{\text{new}}]$ is a new vertex added to G, with the cost function (cost($[\mathbf{s}_{\text{new}}]$)) associated with the distance from $[\mathbf{s}_{\text{init}}]$ to $[\mathbf{s}_{\text{new}}]$ through its parent $[\mathbf{s}_{\text{nearest}}]$. Next, the *near* procedure checks if a better parent for $[\mathbf{s}_{\text{new}}]$ can be found. Therefore, a list of potential vertices in a neighborhood $[\mathbf{s}_{\text{near}}] \in \mathscr{S}_{\text{near}}$ of $[\mathbf{s}_{\text{new}}]$ is selected. For each vertex ($[\mathbf{s}_{\text{near}}]$) from the list of potential parents $\mathscr{S}_{\text{near}}$ is checked if the cost (according to the distance metric as defined in (3)) to arrive in $[\mathbf{s}_{\text{new}}]$ through $[\mathbf{s}_{\text{near}}]$ is better than cost($[\mathbf{s}_{\text{new}}]$). If it is the case and the path is collision free then the *rewire-parent* procedure will update $[\mathbf{s}_{\text{new}}]$ parent with $[\mathbf{s}_{\text{near}}]$ and its cost($[\mathbf{s}_{\text{new}}]$) accordingly. These steps (Lines 4 to 13) are repeated until the algorithm reaches the MaxIter iterations or until $[\mathbf{s}_{\text{new}}] = [\mathbf{s}_{\text{goal}}]$, or more likely when $[\mathbf{s}_{\text{new}}] \subset [\mathbf{s}_{\text{goal}}]$. Note that here as in Subsect. 3.2 $[\mathbf{s}_{\text{init}}] = \text{Hull}(\mathbb{S}_{\text{init}})$, $[\mathbf{s}_{\text{obs}}] = \text{Hull}(\mathbb{S}_{\text{obs}})$ and $[\mathbf{s}_{\text{goal}}] = \text{Int}(\mathbb{S}_{\text{goal}})$.

Random-box-GoalBias, ***Nearest-neighbor*** and ***Collision-free-path*** are already described in Section 3.2.

Steer procedure*:** The control input used by this procedure is the same as the one proposed for the ***sciBoxRRT motion planner, i.e. designed according to a desired behaviour or to a chosen criterion.

***Near procedure*:** In this study, the k-nearest neighbors algorithm is employed to determine the set of vertices nearest to the state $[\mathbf{s}_{\text{new}}]$, according to the metric d, defined in (3). At each iteration, $\mathscr{S}_{\text{near}}$ will contain the closest vertices with the metric $d([\mathbf{s}_{\text{new}}], G) < r$. This means that the vertices contained in $\mathscr{S}_{\text{near}}$ are searched within the area of a ball of radius $r(n) = \gamma \log(n)$, with $\gamma > \varepsilon(1 + \frac{1}{\text{dim}})$ or $\gamma = 2\varepsilon$ as suggested in [9] where ε is Euler's number, n is the number of vertices in the tree at a given iter-

input : $[\mathbf{s}_{\text{init}}] \subset \mathbb{S}_{\text{free}}$, $[\mathbf{s}_{\text{goal}}] \subset \mathbb{S}_{\text{free}}$, MaxIter $\in \mathbb{N}$
output: G

$G.init([\mathbf{s}_{\text{init}}])$;
i $\leftarrow 0$;
repeat
  $[\mathbf{s}_{\text{rand}}] \leftarrow$ random-box-*GoalBias* $(\mathbb{S}_{\text{free}})$;
  $[\mathbf{s}_{\text{nearest}}] \leftarrow$ nearest-neighbor$(G, [\mathbf{s}_{\text{rand}}])$;
  $([\mathbf{s}_{\text{new}}], \mathbf{u}) \leftarrow$ steer$([\mathbf{s}_{\text{nearest}}], [\mathbf{s}_{\text{rand}}])$;
  if *collision-free-path(*$[\mathbf{s}_{nearest}], [\mathbf{s}_{new}]$*)* **then**
    cost$([\mathbf{s}_{\text{new}}]) \leftarrow$ cost$([\mathbf{s}_{\text{nearest}}])$ + d$([\mathbf{s}_{\text{nearest}}], [\mathbf{s}_{\text{new}}])$;
    $\mathscr{S}_{\text{near}} \leftarrow$ near$(G, [\mathbf{s}_{\text{new}}])$;
    $([\mathbf{s}_{\text{near}}], \mathbf{u}) \leftarrow$ NewParent$(\mathscr{S}_{\text{near}}, [\mathbf{s}_{\text{nearest}}], [\mathbf{s}_{\text{new}}])$;
    $G \leftarrow$ Rewire-Parent$([\mathbf{s}_{\text{near}}], [\mathbf{s}_{\text{new}}], G)$;
    return $[\mathbf{s}_{\text{new}}]$
  return $\emptyset$
until $(i++ < MaxIter)$ **or** *(*$[\mathbf{s}_{new}] \neq \emptyset$ **and** $[\mathbf{s}_{new}] \subset [\mathbf{s}_{goal}]$*)*;
return G

Algorithm 2: Towards BoxRRT* motion planning algorithm

ation and dim represents the dimension of the configuration space.
***NewParent procedure*:** When a vertex is added in G its cost is defined as: cost$([\mathbf{s}_{\text{new}}]) =$ cost$([\mathbf{s}_{\text{nearest}}]) + d([\mathbf{s}_{\text{nearest}}], [\mathbf{s}_{\text{new}}])$, where cost$([\mathbf{s}_{\text{nearest}}])$ represents the distance from the initial state $([\mathbf{s}_{\text{init}}])$ to the vertex initial parent $[\mathbf{s}_{\text{nearest}}]$. This procedure verifies if among the vertices in $\mathscr{S}_{\text{near}}$ a better parent can be found. For each vertex $[\mathbf{s}_{\text{near}}] \in \mathscr{S}_{\text{near}}$ it is checked if the total cost to arrive to $[\mathbf{s}_{\text{new}}]$, passing through $[\mathbf{s}_{\text{near}}]$ is smaller than cost$([\mathbf{s}_{\text{new}}])$. When a better parent for $[\mathbf{s}_{\text{new}}]$ is found, the steer procedure is applied from $[\mathbf{s}_{\text{near}}]$ to $[\mathbf{s}_{\text{new}}]$. The control input applied can drive $[\mathbf{s}_{\text{near}}]$: (a) to $[\mathbf{s}_{\text{new}}]$ state, (b) inside $[\mathbf{s}_{\text{new}}]$ state or (c) as close as possible and with as small cost to $[\mathbf{s}_{\text{new}}]$.
***Rewire-Parent procedure*:** If a better parent is found along with a control input which connects it to $[\mathbf{s}_{\text{new}}]$, by the *NewParent* procedure, the *Rewire-Parent* procedure will update $[\mathbf{s}_{\text{new}}]$ parent and cost value.

Even though the ***tBoxRRT**** motion planner is build upon the original RRT* planner principles, the rewire procedure, now denoted *Rewire-Parent*, searches only potential parents and not potential children, as done in the original RRT* planner. Moreover, the original RRT* planner stops the algorithm when a given number of iterations is reached, while as seen in Algo. 2 our proposed planner stops when a solution is found. Even if the choice of the stop criteria can have an impact in proving a near-optimal solution for the proposed algorithm, the latter benefits of different advantages such as guaranteeing the reliability and robustness of the found solution, if exists.

5 Numerical Example

The three proposed motion planners are applied on four different scenarios for which the configuration space size is 0.6m $\times$0.6m$\times 2\pi$rad. The initial state $[\mathbf{s}_{\text{init}}]$ size for each environment is: 0.2m $\times$ 0.2m $\times$ 0.02rad, 0.3m $\times$ 0.3m $\times$ 0.02rad, 0.4m $\times$ 0.4m $\times$ 0.02rad, 0.2m $\times$ 0.2m $\times$ 0.02rad and has to reach the following final state $[\mathbf{s}_{\text{goal}}]$ size: 2m $\times$ 2m $\times$ πrad, 3m $\times$ 3m $\times$ πrad, 3m $\times$ 3m $\times$ πrad, $6m \times 6m \times \pi$rad.

Each algorithm performs 50 iterations for each proposed scenario on a Intel Core m7-6Y75 CPU at 1.20GHz×4. The used software consists in DynIBEX[1] [5] which is a library providing operators to deal with constraint satisfaction problems embedding differential equations.

5.1 Robot mobile modelling

The considered mobile robot is represented by a simple car model, evolving in a 2D environment. The car moves in a configuration $\mathbf{s} = (x\ y\ \theta)$ with its position $(x\ y)$ and orientation θ w.r.t. a frame attached to the environment. The simple car model which involves non-holonomic constraints is as follows:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \frac{v}{L}\tan(\delta), \tag{6}$$

where the control input $u = [v\ \delta]$ is represented by the longitudinal speed $v \in [-1\ 1]$ and the steering angle $\delta \in [-\frac{\pi}{2}\ \frac{\pi}{2}]$. $L = 1.5$ represents the distance between the front and back axes of the car.

While the control input employed by the ***rciBoxRRT*** planner is randomly chosen in the admissible set, the one used by ***sciBoxRRT*** and ***tBoxRRT**** planners is designed in two steps, as follows. First, the car is oriented towards the target, case in which the control input is obtained by considering the error between the orientation and direction to the goal equal to zero: $\text{atan}(\frac{y_{\text{target}}-y_{\text{start}}}{x_{\text{target}}-x_{\text{start}}}) - \theta_{\text{target}} = 0$, and using it to simulate (6) for Δt. Once the car is oriented towards the target a second control input is designed to move the car straight ahead to the target. The steering angle is equal to 0, while the longitudinal speed is obtained from: $v = \frac{\dot{x}}{\sin(\theta)}$. Finally, (6) is simulated for Δt with this last control input.

5.2 Simulations

All scenarios are performed with $\Delta t = 1$s, a probability $p = 0.33$ mentioned in Section 3.1 and the maximum limit of iterations fixed to MaxIter $= 20.000$. The three proposed motion planners are performed on four different environments denoted env i with $i = 1..4$ and illustrated on Fig. 3. On the same figure a solution found by each algorithm is represented along with the exhibited total number of vertices and the computation time (CPU [s]). When the complexity of the environment increase, as well the algorithm's performances in terms of CPU, number of vertices and distance for the planned path will increase.

Fig. 1 reports the number of iterations necessary for the convergence of each algorithm. We observe that ***rciBoxRRT*** which applies a random control input requires the most iterations for convergence, while ***sciBoxRRT*** and ***tBoxRRT**** which use a designed control, presented above, converged after less iterations. Fig. 2 illustrates the mean and standard deviation of computational time, number of vertices and length of the planned path, for all simulations performed by the three planners. For all environments, as Fig. 2 stands for, while comparing the planners two different classifications can be made which is the same for all 4 environments: (a) in terms of CPU time, the order of the planners performance enumerated from the more expensive to the less one is: ***rciBoxRRT***, ***tBoxRRT**** and ***sciBoxRRT***; (b) in terms of number of vertices and length of the planned path, the order of the planners performances given in decreasing order is: ***rciBoxRRT***, ***sciBoxRRT*** and ***tBoxRRT****.

[1] http://perso.ensta-paristech.fr/∼chapoutot/dynibex/

These results suggest that the two planners for which a control input is designed have better performances than the one in which a random control input is used. Moreover, it was not a surprise to see that ***tBoxRRT**** is more time consuming than ***sci-BoxRRT*** while the first one recalls multiples times the *steer* procedure (Lines 10-11 in Algo. 2) so that better length path performances to be obtained.

6 Conclusion and Perspectives

Improved versions of the previously proposed boxRRT algorithm and a new motion planner tBoxRRT* based on RRT* are presented in this paper. All motion planners are able to find reliable and robust paths in an uncertain environment, where the uncertain quantities are assumed to belong to boxes.

If the imperfections on the initial states are too large, the imprecision at each new uncertain state can increase. This issue, which will be the topic of future studies, can be encountered by our proposed planners and by the original BoxRRT planner as well. For this reason, in practical settings the motion planners can be updated from time to time by using observers to estimate the state evolution using informations provided by sensors. This can be very useful in decreasing the large imperfections of these uncertain new states. Also, the use of different control inputs between two states can limite the uncertain new states growth.

In the presented version, the ***tBoxRRT**** has a basic form in which not all the RRT* principles are employed (in particular the stop criteria and the rewire procedure). Applying those principles would improve the planned path length value. The proposed planners can be adapted for cases where the free subspace of the configuration space varies with time, to describe moving obstacles. Moreover, model or/and sensorial uncertainties along with a more complex model of the mobile robot which takes into account its dynamics can be considered. This should form the subject of future studies.

Acknowledgment

The authors would like to thank Oliver Mullier, Julien Alexandre dit Sandretto, Eric Goubault and Benjamin Martin for useful comments and discussions.

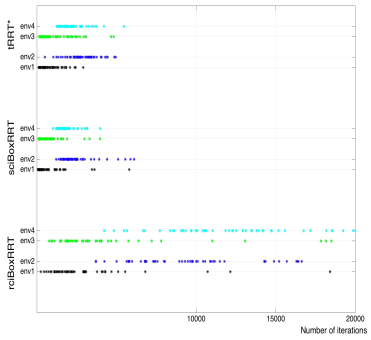
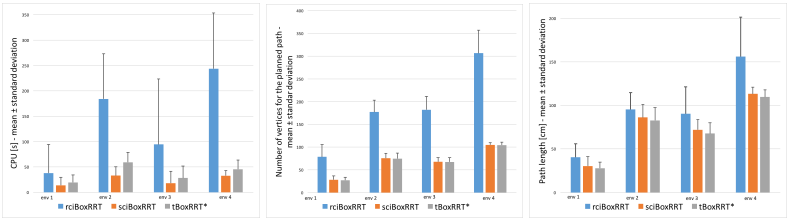


Figure 1: Number of iteration for convergence required by each of the three proposed algorithms.



(a) Computational time (s) required by the three proposed algorithms for convergence

(b) Number of vertices for the planned path obtained by the three proposed algorithms

(c) Planned path length (cm) obtained by the three proposed algorithms.

Figure 2: The ***rciBoxRRT***, ***sciBoxRRT*** and ***tBoxRRT**** path planner found performances for each of the four proposed scenarios.

References

[1] J. Alexandre dit Sandretto and A. Chapoutot. Validated explicit and implicit Runge-Kutta methods. *Reliable Computing*, 2016.

[2] J. Alexandre dit Sandretto and A. Chapoutot. Validated simulation of differential algebraic equations. *Reliable Computing*, 2016.

[3] J. Alexandre dit Sandretto, A. Chapoutot, and O. Mullier. Formal Verification of Robotic Behaviors in Presence of Bounded Uncertainties. In *Conference on Robotic Computation*. IEEE, 2017.

[4] A. Censi, D. Calisi, A. De Luca, and G. Oriolo. A bayesian framework for optimal motion planning with uncertainty. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1798–1805, May 2008.

[5] J. Alexandre dit Sandretto and A. Chapoutot. DynIBEX: a differential constraint library for studying dynamical systems (poster). In *Conference on Hybrid Systems: Computation and Control*. ACM, 2016.

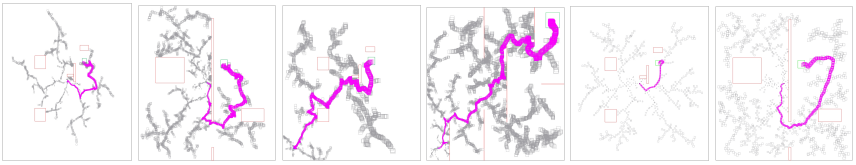
[6] L. Jaulin. Path planning using intervals and graphs. *Reliable Computing*, 7(1):1–15, fevrier 2001.

[7] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer-Verlag, 2001.

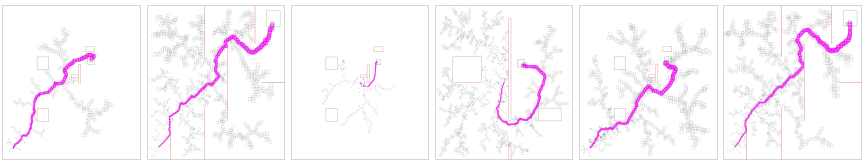
[8] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Conference on Decision and Control*, pages 7681–7687. IEEE, 2010.

[9] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[10] J. J. Jr Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Conference on Robotics and Automation*, volume 2, pages 995–1001. IEEE, 2000.



(a) **rciBoxRRT** env 1 (b) **rciBoxRRT** env 2 (c) **rciBoxRRT** env 3 (d) **rciBoxRRT** env 4 (e) **sciBoxRRT** env 1 (f) **sciBoxRRT** env 2



(g) **sciBoxRRT** env 3 (h) **sciBoxRRT** env 4 (i) **tBoxRRT*** env 1 (j) **tBoxRRT*** env 2 (k) **tBoxRRT*** env 3 (l) **tBoxRRT*** env 4

Figure 3: **(a) - (d)** ***rciBoxRRT*** (**(a)** total vertices 2200 in 28 [s]; **(b)** total vertices 5880 in 103 [s]; **(c)** total vertices 3416 in 51 [s]; **(d)** total vertices 7802 in 141[s]). **(e) - (h)** ***sciBoxRRT***(**(e)** total vertices 570 in 11 [s]; **(f)** total vertices 1149 in 32 [s]; **(g)** total vertices 278 in 5[s]; **(h)** total vertices 978 in 26[s]). **(i) - (l)** ***tBoxRRT****(**(i)** total vertices 156 in 3 [s]; **(j)** total vertices 1088 in 38 [s]; **(k)** total vertices 786 in 20 [s]; **(l)** total vertices 963 in 28[s]).

[11] A. Lambert and D. Gruyer. Safe path planning in an uncertain-configuration space. *Conference on Robotics and Automation*, 2003.

[12] S. M. LaValle. Rapidly-exploring random trees: a new tool for path planning. Technical report, Iowa State University, 1998.

[13] S. M. LaValle. *Planning Algorithms*. Cambridge Univ. Press, 2006.

[14] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics*, 2000.

[15] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[16] N.S. Nedialkov, K.R. Jackson, and G.F. Corliss. validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105:21–68, 1999.

[17] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[18] I. Noreen, A. Khan, and Z. Habib. Optimal path planning using RRT* based approaches: A survey and future directions. *Advanced Computer Science and Applications*, 7(11), 2016.

[19] L. A. Page and A. C. Sanderson. Robot motion planning for sensor-based control with uncertainties. In *Int. Conf. Robotics and Automation*, volume 2, pages 1333–1340 vol.2, May 1995.

[20] R. Pepy, M. Kieffer, and E. Walter. Reliable robust path planner. In *Int. Conf. Intelligent Robots and Systems*. IEEE, 2008.

[21] R. Pepy, M. Kieffer, and E. Walter. Reliable robust path planning with application to mobile robots. *Int. J. Appl. Math. Comput. Sci.*, 19(3):413 – 424, 2009.

[22] R. Pepy and A. Lambert. Safe path planning in an uncertain-configuration space using rrt. In *Int. Conf. Intelligent Robots and Systems*, pages 5376–5381. IEEE, 2006.

[23] P. Pharpatara, B. Hérissé, and Y. Bestaoui. 3-d trajectory planning of aerial vehicles using RRT*. *Trans. on Control Systems Technology*, PP(99), 2016.

[24] P. Pharpatara, B. Hérissé, R. Pepy, and Y. Bestaoui. Shortest path for aerial vehicles in heterogeneous environment using RRT*. In *International Conference on Robotics and Automation*. IEEE, 2015.