Robust Motion Planning Based on Sliding Horizon and Validated Simulation *

Elliot Brendel, Julien Alexandre Dit Sandretto, Alexandre Chapoutot

Robust Motion Planning Based on Sliding Horizon and Validated Simulation*

Elliot Brendel, Julien Alexandre dit Sandretto, and Alexandre Chapoutot †

March 21, 2017

Abstract

A new algorithm of motion planning based on set-membership approach and inspired by optimal control methods is presented. The goal of this algorithm is to find a safe and optimal path taking into account various sources of uncertainties on the dynamical model of the plant, on the model of the environment, while being robust with respect to the numerical approximations introduced by numerical integration methods. The main approach is based on sliding horizon method in order to predict the behavior of the plant allowing the computation of an optimal path. As an example, the motion planning algorithm is applied on an Autonomous Underwater Vehicle (AUV) case study, showing the benefit of the proposed approach.

1 Introduction

Motion planning algorithms are a center piece in the control framework of mobile robots as they contribute to give them the ability of have autonomous behaviors. Furthermore, such class of algorithms is critical as a failure can cause the abort of the mission or can cause important amount of damage such as human loss. The validation of such algorithms is then mandatory in order to increase the confidence of the end users. However, those algorithms are also subject to constraints, *e.g.*, to reduce fuel consumption. In consequence, computing a safe path is usually not enough, an optimal one is search to minimize some costs.

Moreover, one of the challenge in order to design robust and reliable motion planning algorithms is to take into account various sources of uncertainties. For example, the environment is not exactly know and some disturbance should have been considered. Mathematical models of the mobile robots are not perfect and usually come from some simplification in order to have efficient simulation activities. Lastly, computer-aided design usually produces approximated results as it is based on numerical methods which cannot produce close form solution of a problems, *e.g.*, the solution of an initial value problem for ordinary differential equations. The set-membership framework is suitable to deal with such kinds of uncertainties.

*This work benefited from the support of the "Chair Complex Systems Engineering - Ecole Polytechnique, THALES, DGA, FX, DASSAULT AVIATION, DCNS Research, ENSTA ParisTech, Télécom ParisTech, Fondation ParisTech and FDO ENSTA".

†Authors are with ENSTA ParisTech, Université Paris-Saclay, 828, bd des maréchaux, 91762 Palaiseau Cedex, France {brendel,alexandre, chapoutot}@ensta.fr

Contributions The main contribution of this article is the combination of set-membership methods with optimizing approach. Hence, a correct-by-construction algorithm is defined with the intrinsic properties to be robust to uncertainties as it relies on set-membership approach [8]. Moreover, embedding the motion planning problem into a constraint satisfaction problems (CSP) [17], and more precisely into a global optimization framework [5], the proposed algorithm produces an optimal free-collision paths with respect to a given cost function which is minimized.

Related Work Motion planning is an active research area in Robotics and many methods have been developed. Artificial potential fields has been used in motion planning problems such as in [3] but without producing optimal path. A popular approach is to use stochastic sampling to discretize the configuration space, *e.g.*, the *Rapidly-exploring Random Trees* (RRT) [11] path planning algorithm and its many variants. The (asymptotic) optimality of the solution is provided by the optimal Rapidly-exploring Random Trees (RRT*) first proposed in [9]. Other methods based on receding horizon approach have also been considered to produce optimal collision-free paths. For example, [12] uses receding horizon in context of multi-robot or [18] considers complex mission described by temporal logic. While all these methods are efficient to produce collision-free paths, they usually did not take into account uncertainties and so the robustness of the solution is not guaranteed.

Uncertainty in motion planning has been considered mainly based on two representations: set-membership [15, 16] or covariance matrices [10, 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. In [16] is introduced a preliminary conceptual reliable and robust path planner based on RRT principles and solved in an set-membership framework where all uncertainties are considered bounded. Interval analysis principle along with graph algorithms were previously used [6] to find a collision-free shortest path for a polygonal rigid object in a given configuration with obstacles. Nevertheless, theses approaches builds robust and safe paths but usually cannot produce optimal paths.

Contents The paper is organized as follows. Some preliminary notions on motion planning problem, set-membership methods and sliding horizon method are introduced in Section 2. The main contribution of the paper is presented in Section 3 by formulated the problem and the presentation of the algorithms. In Section 4 a case study focus on AUV is described showing the relevance of the proposed approach. Conclusion and perspective are drawn in Section 5

2 Preliminary notions

In this section, the main notions useful for our approach to solve the robust motion planning problem are introduced.

2.1 Dynamics of a vehicle

The dynamics of a vehicle, such as a car, a flight or a ship, can be modelized by differential equations to analyze its behavior. In the special case of autonomous vehicles - Unmanned Aerial Vehicle (UAV), Autonomous Underwater Vehicle (AUV) or Unmanned Ground Vehicle (UGV) - the dynamical modeling is very important to define

the controller and the path planner. In this paper, it is assumed that the dynamics is modeled by nonlinear ordinary differential equations as the ones coming from the Newton's laws.

Starting from a given point at time zero, an initial value problem is defined by:

$$\dot{\mathbf{y}}(t) = f(t, \mathbf{y}(t), \mathbf{u}(t)), \text{ with } \mathbf{y}(0) = \mathbf{y}_0 \ . \tag{1}$$

In (1), $\mathbf{y}(t) \in \mathbb{R}^n$ and $\mathbf{u}(t) \in \mathbb{R}^m$ denote the vector of states and inputs, respectively.

2.2 Motion planning

A motion planning consists on producing a continuous path in the configuration space, that satisfies system dynamics (movement constraints), safety constraints (obstacles), inputs limitations, and possibly optimizes a cost linked to a given aspect of movement. In a more enriched motion planning, the considered constraints can take into account the quality of ground, ocean stream, up-draft, seabed shape, etc. These constraints can be considered in connection with the task and/or the sensors. A cost function are often added. This latter, linked to states of the system (and not linked to inputs) can be a distance expressed in the configuration space or a more complex functional cost involving state and/or state derivatives.

2.3 Uncertainties and Validated Simulation

The simplest and most common way to represent and manipulate sets of values is *interval arithmetic* (see [13]). An interval $[x_i] = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i such that $\underline{x_i} \leq x_i \leq \overline{x_i}$. $\mathbb{IR}$ denotes the set of all intervals over reals.

Interval arithmetic extends to $\mathbb{IR}$ elementary functions over $\mathbb{R}$. For instance, the interval sum, *i.e.*, $[x_1] + [x_2] = [\underline{x_1} + \underline{x_2}, \overline{x_1} + \overline{x_2}]$, encloses the image of the sum function over its arguments. An interval vector or a *box* $[\mathbf{x}] \in \mathbb{IR}^n$, is a Cartesian product of n intervals.

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 $\mathbf{y}_0, \ldots, \mathbf{y}_{\text{end}}$, by the help of a guaranteed integration scheme. In details, a guaranteed integration scheme is made of

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

A validated numerical integration method is a two step method starting at time t_j and for which *i*) it computes an enclosure $[\tilde{\mathbf{y}}_j]$ of the solution of (1) over the time interval $[t_j, t_{j+1}]$ to bound $\text{lte}_\Phi(f, \mathbf{y}_j, t_j, h)$; *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 [14, 1] and the references therein for more details.

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

$$R : \begin{cases} \mathbb{R} \rightarrow \mathbb{IR}^n \\ t \mapsto [\mathbf{y}] \end{cases} \tag{2}$$

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

$$\tilde{R} : \begin{cases} \mathbb{IR} \rightarrow \mathbb{IR}^n \\ [\underline{t},\overline{t}] \mapsto [\tilde{\mathbf{y}}] \end{cases} \tag{3}$$

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

In [2], validated numerical integration has been combined with CSP tools. Indeed, functions $R()$ and $\tilde{R}()$ are abstracted, but guaranteed, solutions of (1). Therefore, the process of validated simulation, mixed with constraint programming and abstraction of time functions provides an efficient tool for the prediction of the system evolution.

2.4 Sliding Horizon Method

A sliding, or receding, horizon-based method is used when the future cannot be anticipated, such as in a moving environment or a progressively discovered environment.

In the proposed method, the predicted horizon needs to be prolonged, but long term validated simulations are difficult to obtain. A sliding horizon approach is then interesting to progressively discover the environment and by the way reducing the duration of the simulations. A sliding horizon based motion planning is then close to Model Predictive Control. The main difference is the desired result: MPC is a control synthesis, then it produces solutions in the input space (with cost on inputs), while motion planning is a path planner, then it produces solutions in the state space (with cost on states).

A sliding horizon-based method consists on computing a value (an optimal control or a path) from a time t_0 for a period T_0 (as long as possible) in order to anticipate the future, *i.e.*, the horizon, injecting this value in the system, and re-computing this value at time t_1, with $t_0 + t_1 < T_0$, for a new horizon period T_1 (T_1 may be equal to T_0). The initial condition of the problem on t_1 is the state of the system evaluated or measured (in the case of control synthesis) at t_1.

3 Robust and Guaranteed Motion Planning Based on Sliding Horizon

3.1 Problem Formulation

A dynamical system as defined in (1), where $\mathbf{y}$ is the state and $\mathbf{u}$ is the control, is considered. Then a constraint function $h : \mathbf{y} \mapsto h(\mathbf{y})$ and a cost function $g : \mathbf{y} \mapsto g(\mathbf{y})$ are defined, constraints coming from safety and cost from task for example. Thus, the problem is to find the control $\mathbf{u}$ which is the solution of

$$(P) : \begin{cases} \min_{\mathbf{u}} \quad g(\mathbf{y}) \\ \dot{\mathbf{y}} = f(\mathbf{y},\mathbf{u}) \\ h(\mathbf{y}) < 0 \end{cases} .$$

3.2 Complete Approach

The robust and guaranteed motion planning using sliding horizon proposed in this paper is defined by the steps:

1. a control $\mathbf{u}_1$ solution of (P) on a time span $[0,T]$, with T the prediction period, and from the initial state $\mathbf{y}_0$, is computed;
2. a simulation on a time span $[0,T']$, with T' the sliding period $(T' < T)$, and from the initial state $\mathbf{y}_0$, is performed. It provides the trajectory $\mathbf{y}_1(t)$ driven by the input $\mathbf{u}_1$;
3. a control $\mathbf{u}_2$ solution of (P) on a time span $[T', T'+T]$, and from the initial state $\mathbf{y}_1(T')$, is computed;
4. as in the second step, a simulation on a time span $[T', 2T']$, and from the initial state $\mathbf{y}_1(T')$ is performed. It provides the trajectory $\mathbf{y}_2(t)$ driven by the input $\mathbf{u}_2$;
5. these steps are repeated until a user defined number of steps is reached.

At Step i, to find the control $\mathbf{u}_i$, we browse through the set of the controls, assuming that its finite, until the control whose corresponding $\mathbf{y}(t)$ satisfies the constraint $h(\mathbf{y}(t)) < 0$ and minimizes the cost $g(\mathbf{y}(t))$, $\forall t \in [(i-1)T', iT']$ is found.

3.3 Algorithms

The method is divided in two algorithms, the main one (see Algorithm 1) is the global procedure handling the sliding horizon progress. The second one (see Algorithm 2) solves the problem (P). A third algorithm (see Algorithm 3) which improved the second one is given.

3.3.1 Main Algorithm

Algorithm 1 Sliding horizon

Require: $\mathbf{y}_0$, T, T', n_{step}, f, h, g, D
  for $i \in [\![0, n_{\text{step}} - 1]\!]$ **do**
    $\mathbf{u}_{\text{optim}} \longleftarrow$ find_u $(\mathbf{y}_0, T, f, h, g, D)$
    $\mathbf{y}(t) \longleftarrow$ simulation$\big(f, \mathbf{y}_0, \mathbf{u}_{\text{optim}}, T'\big)$
    $\mathbf{y}_0 \longleftarrow \mathbf{y}(T')$
    Path.push_back$(\mathbf{y}(t))$
  end for
Ensure: Path

In Algorithm 1, n_{step} is the iteration number of the algorithm and $\mathbf{y}_0$ is the initial state for an iteration. h and g are the constraint function and the cost function, respectively. T' and T are the same than given in Section 3.2 and D is the set of inputs which is assumed to be finite. The function find_u solves our problem (P) (and is given in Algorithm 2). The simulation method returns a list of boxes that contains the solution of $\dot{\mathbf{y}} = f(\mathbf{y}, \mathbf{u})$ for a given $\mathbf{u}$ in a guaranteed way using validated simulation method (see Section 2.3). Path is the list of trajectories from the initial state driven by the inputs found for every iteration of the algorithm.

3.3.2 Optimization Algorithm

In order to find a control which is solution of (P), a discretization of the set of the controls is performed, and the one which minimizes g while satisfying the constraint h is returned. In Algorithm 2, D is a finite set of control inputs, $\mathbf{u}_{\text{temp}}$ is the previous

Algorithm 2 find_u function

Require: $\mathbf{y}_0$, T, f, h, g, D
  $c \longleftarrow +\infty$
  for $\mathbf{u} \in D$ **do**
    $\mathbf{y}(t) \longleftarrow \text{simulation}(f, \mathbf{y}_0, \mathbf{u}, T)$
    if $g(\mathbf{y}(T)) < c$ **and** $h(\mathbf{y}(t)) < 0, \forall t \in [0,T]$ **then**
      $\mathbf{u}_{\text{temp}} \longleftarrow \mathbf{u}$
      $c \longleftarrow g(\mathbf{y}(T))$
    end if
  end for
  $\mathbf{u}_{\text{optim}} \longleftarrow \mathbf{u}_{\text{temp}}$
Ensure: Optimal control $\mathbf{u}_{\text{optim}}$

validated control, and c is the cost corresponding to $\mathbf{u}_{\text{temp}}$. The result is given in term of the "optimal control" $\mathbf{u}_{\text{optim}}$, which is proven to drive the system to the optimal and safe trajectory, while being easier to save in memory.

3.3.3 Improvement of Algorithm 2

Algorithm 3 Improvement of Algorithm 2: find_u function

Require: $\mathbf{y}_0$, T, f, h, g, D
  $c \longleftarrow +\infty$
  $D' \longleftarrow D$
  for $\mathbf{u} \in D'$ **do**
    $\mathbf{y}(t) \longleftarrow \text{simulation}(f, \mathbf{y}_0, \mathbf{u}, T)$
    if $g(\mathbf{y}(T)) < c$ **and** $h(\mathbf{y}(t)) < 0, \forall t \in [0,T]$ **then**
      $\mathbf{u}_{\text{temp}} \longleftarrow \mathbf{u}$
      $c \longleftarrow g(\mathbf{y}(T))$
    end if
    $D' \longleftarrow D' \setminus \{\mathbf{u}\}$
    $\mathbf{y}_{D'}(t) \longleftarrow \text{simulation}(f, \mathbf{y}_0, D', T)$
    if $c \leqslant g(\mathbf{y}_{D'}(T))$ **then**
      break
    end if
  end for
  $\mathbf{u}_{\text{optim}} \longleftarrow \mathbf{u}_{\text{temp}}$
Ensure: Optimal control $\mathbf{u}_{\text{optim}}$

In this part, an improvement of the previous algorithm using the set-membership approach is proposed. For each control $\mathbf{u}$ found, considering that D' is the set of the remaining untested controls, a simulation on all the set D' can be performed by the help of validated numerical integration methods (see Section 2.3). We denote by $\mathbf{y}_{D'}(t)$ the

obtained trajectories. If $c \leqslant g(\mathbf{y}_{D'}(T))$, we ensured that the previous validated control $\mathbf{u}$ is optimal and no more iteration over D is needed.

4 Motion Planning For an AUV

The motion planning of an AUV which has to move the closest to the seabed is considered. In consequence, the cost function is the depth of the gravity center of the AUV. As security constraints, we want to ensure that the AUV is closer to the seabed than the distance $d_{\max}$ and further than $d_{\min}$.

4.1 Dynamics of an AUV

The gravity center of the AUV is subjected to the ODE defined in [7] and given in Equation (4).

$$\begin{cases} \dot{x} &= v\cos\theta\cos\psi \\ \dot{y} &= v\cos\theta\sin\psi \\ \dot{z} &= -v\sin\theta \\ \dot{\psi} &= \frac{\sin\varphi}{\cos\theta}\cdot v\cdot u_1 + \frac{\cos\varphi}{\cos\theta}\cdot v\cdot u_2 \\ \dot{\theta} &= \cos\varphi\cdot v\cdot u_1 - \sin\varphi\cdot v\cdot u_2 \\ \dot{\varphi} &= -0.1\sin\varphi + \theta\cdot v\cdot(\sin\varphi\cdot u_1 + \cos\varphi\cdot u_2) \end{cases} \tag{4}$$

where $\mathbf{s} = (x, y, z, \psi, \theta, \varphi)$ is the state vector which can be split into the vector (x, y, z) of the coordinates of the gravity center and the vector (ψ, θ, φ) of the Euler angles; $\mathbf{u} = (u_1, u_2)$ is the control input vector; v is the velocity.

Note that (4) has been simplified by substituting $\tan\theta$ by θ in the definition of $\dot{\varphi}$ to avoid technical issues of the implementation. Nonetheless, the algorithm remains valid.

4.2 Underwater environment

We define a function $(x, y) \mapsto \text{seabed}(x, y)$ which returns the depth of the seabed at the coordinates (x, y). We also define $d_{\min}$ and $d_{\max}$ two constants such that the AUV stays at a distance to the seabed between $d_{\min}$ and $d_{\max}$. Some constraints on AUV angles are considered: yaw and roll are bounded in an interval (to go in a quite straight way and to not capsize) and pitch is bounded by an extreme value (to limit the dive angle). Finally, in order to force the AUV to move forward through the x dimension, we impose $x_{\text{end}} > x_{\text{init}}$. Thus, the problem is to find the control $\mathbf{u}$ solution of

$$(P_{\text{AUV}}) : \begin{cases} \min_{\mathbf{u}} \quad z \\ \dot{\mathbf{s}} = f(\mathbf{s}, \mathbf{u}) \\ z > \text{seabed}(x, y) + d_{\min} \\ z < \text{seabed}(x, y) + d_{\max} \\ x_{\text{end}} > x_{\text{init}} \\ \theta < 0.8 \\ \varphi, \psi \in [-0.5, 0.5] \end{cases}$$

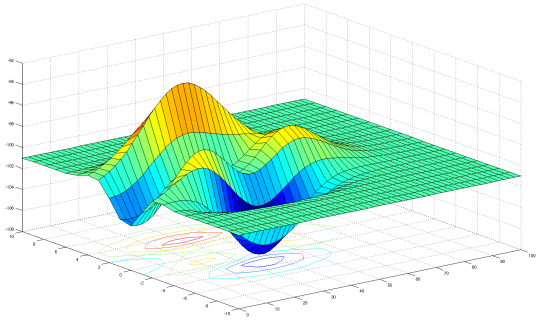

Figure 1: Seabed picture.

4.3 Experiments

4.3.1 Settings of the experiments

The **seabed function** is defined by

$$\begin{array}{rcl} \text{seabed} : (x,y) & \mapsto & 3(1-x)^2 e^{-x^2-(y+1)^2} \\ & & -10\left(\frac{x}{5}-x^3-y^5\right)e^{-x^2-y^2} \\ & & -\frac{e^{-(x+1)^2-y^2}}{3} \end{array}$$

Then the following seabeds are considered

- seabed 1: $(x,y) \mapsto \text{seabed}\left(\frac{x-30}{20}, \frac{y}{2}\right) - 100$;
- seabed 2: $(x,y) \mapsto \text{seabed}\left(\frac{y}{2}, \frac{x-30}{20}\right) - 100$;
- seabed 3: $(x,y) \mapsto \text{seabed}\left(\frac{-x+30}{20}, \frac{y}{2}\right) - 100$;
- seabed 4: $(x,y) \mapsto \text{seabed}\left(\frac{y}{2}, \frac{-x+30}{20}, \frac{y}{2}\right) - 100$.

Each seabed is a rotation of the first one (see Figure 1).

The **limits on depth** are defined such that $d_{\min} = 1$ meter $d_{\max} = 10$ meters. The **velocity** v is fixed to 0.1 m.s^{-1} and the **initial state** $\mathbf{s}_0$ to $(0,0,-92,0.1,0.1,0.1)$. As **discretization** of the set of the controls, we choose such that

$$D = \left\{ -0.3 + \frac{0.6k}{9} : k \in [\![0,9]\!] \right\}^2.$$

Finally, the prediction and sliding **periods** are $T = 30$ s and $T' = 15$ s, for a total **number of steps** n_{step} of 35.

4.3.2 Implementation of the find_u function

The presented method has been implemented in the DynIBEX framework[1]. It proposes a validated simulation procedure based on Runge-Kutta methods and provides some differential constraint programming facilities [2]. Every **simulation** has been

Algorithm 4 find_u function for an AUV

Require: $\mathbf{s}_0$, T, f, h

$c \longleftarrow +\infty$
for $i \in [\![0,9]\!]$ **do**
  $u_1 \longleftarrow -0.3 + \frac{0.6i}{9}$
  if not$\big(\mathbf{u}_{\text{temp}}$.is_empty()$\big)$ **then**
    while $i < 10$ **do**
      $\mathbf{u} \longleftarrow (u_1, [-0.3, 0.3])$
      simu_u $\longleftarrow$ simulation$(f, \mathbf{s}_0, \mathbf{u}, T)$
      $\mathbf{s_u} \longleftarrow$ simu_u.get_last()
      $z_{\mathbf{u}} \longleftarrow \mathbf{s}_u[2]$
      if $\text{lb}(c) \leqslant \text{lb}(z_{\mathbf{u}})$ **then**
        $i \longleftarrow i+1$; $u_1 \longleftarrow -0.3 + \frac{0.6i}{9}$
      else
        break
      end if
    end while
    if $i = 10$ **then**
      break
    end if
  end if
  for $j \in [\![0,9]\!]$ **do**
    $u_2 \longleftarrow -0.3 + \frac{0.6j}{9}$; $u \longleftarrow (u_1, u_2)$
    simu_u $\longleftarrow$ simulation$(f, \mathbf{s}_0, \mathbf{u}, T)$
    $\mathbf{s_u} \longleftarrow$ simu_u.get_last()
    $z_{\mathbf{u}} \longleftarrow \mathbf{s_u}[2]$
    if $\text{lb}(z_{\mathbf{u}}) < \text{lb}(c)$ **and** $h(\text{simu\_u}) < 0$ **then**
      $\mathbf{u}_{\text{temp}} \longleftarrow \mathbf{u}$; $\mathbf{s}_{\text{temp}} \longleftarrow \mathbf{s}$; $c \longleftarrow z_{\mathbf{u}}$
    end if
  end for
end for
$\mathbf{u}_{\text{optim}} \longleftarrow \mathbf{u}_{\text{temp}}$

Ensure: Optimal control $\mathbf{u}_{\text{optim}}$

computed using Heun's method with a 10^{-4} precision which is a good trade-off between speed and precision of the results.

Algorithm 4 is an adaptation of the generic Algorithm 3 to the specific problem of the AUV described above and to the dedicated implementation. Some operators coming from DynIBEX are used: lb($[x]$) stands for the lower bound of interval $[x]$, $[x]$.is_empty() corresponds to the test of empty interval, simu_u is the result of a validated numerical simulation method (it contains $R()$ and $\tilde{R}()$ abstractions), in particular simu_u.get_last() returns $R(T)$.

Instead of considering all the elements of the set $D = \left\{-0.3 + \frac{0.6k}{9} : k \in [\![0,9]\!]\right\} \times \left\{-0.3 + \frac{0.6k}{9} : k \in [\![0,9]\!]\right\}$, interval based approach allows us to fix one dimension (here u_1) and to compute the tube of the solutions of (4) for $u_2 = [-0.3, 0.3]$. If this computed tube is less optimal than the previous validated solution $\big(\mathbf{u}_{\text{temp}}, \mathbf{s}_{\text{temp}}\big)$, we skip to the next u_1 until we reach a tube that may contain more optimal solutions than the previ-

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

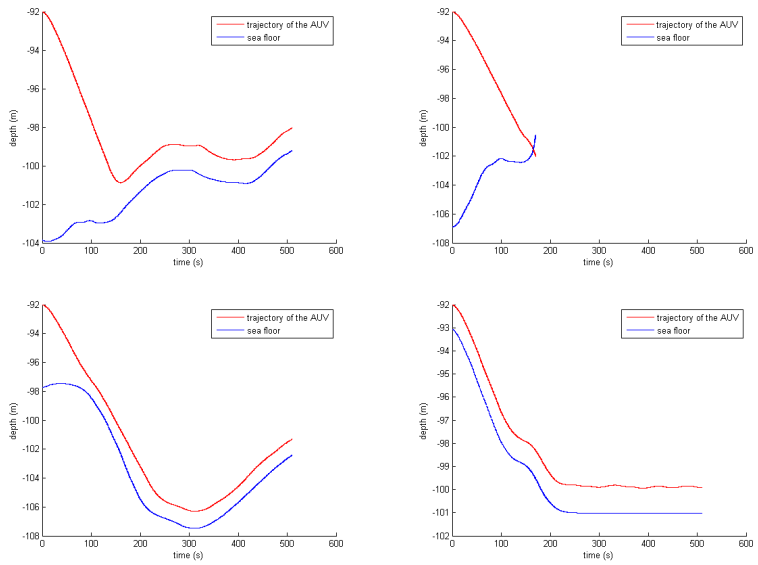

Figure 2: Results of the AUV motion planning with seabed 1 to 4 from top left to bottom right.

ous one (the first break statement). Then we can iterate on the second dimension of the control (u_2) to find a possible new couple $(\mathbf{u}_{\text{temp}}, \mathbf{s}_{\text{temp}})$. The second break statement is reached when there has not been any other tube containing a possible more optimal solution, then we skip all the iterations until the last statement $\mathbf{u}_{\text{optim}} \longleftarrow \mathbf{u}_{\text{temp}}$. With the parameters described in Section 4.3.1, solving the motion planning problem requires 2702 simulations with the improved Algorithm 3 instead of 3557 with Algorithm 2.

4.4 Discussion

Results of the presented method are given in term of depth w.r.t. time in Figure 2. The depth is computed with seabed(x,y). The path provided by the motion planning method for the seabeds 1, 3 and 4 allows the AUV to follow the seabed remaining between a distance $d_{\min}$ and $d_{\max}$ from it. As shown in Figure 2 top right, the execution for the second seabed failed after 9 steps, because of the variations of the seabed (a deep ditch and then a climb). In this picture, after $t = 135$ seconds, a simulation is done with the entire interval of inputs $u = [-0.3, 0.3]^2$, and no solution can be found without collision with the seabed. This phenomenon is due to the fact that the dive velocity of the AUV is high at the moment when the seabed grows quickly.

A solution to this failure of the algorithm could be obtained by increasing T and decreasing T' in order to see the obstacle sooner but with a longer computation time (see Figure 3). An other solution is to add a constraint to the problem, for example we bounded $\text{lb}(\dot{z}) \geqslant -0.03$ m.s^{-1} and the algorithm achieved to find a path across the seabed (see Figure 4). Indeed, it succeeds by avoiding the gap and reaching the optimal depth further.

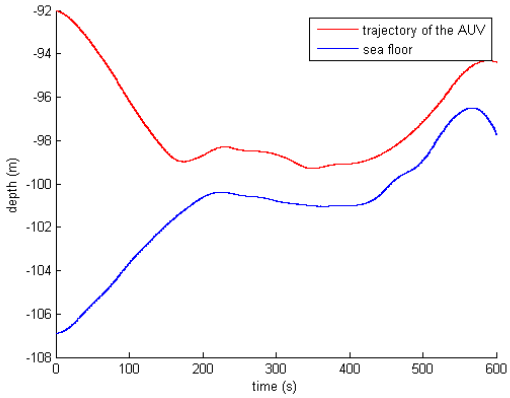

Figure 3: Result for the seabed 2 with longer prediction period.

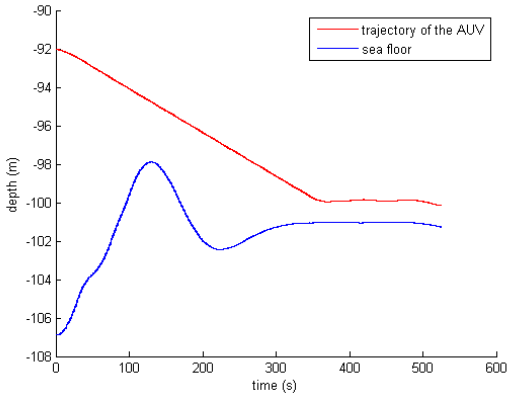

Figure 4: Result for the seabed 2 with additional constraint on diving speed.

5 Conclusion and Future Work

In this paper, we proposed a validated algorithm to solve a motion planning problem, considering constraints and cost on states. Our approach showed its efficiency in an application of motion planning for an Autonomous Underwater Vehicle. An improvement exploiting set membership is used to avoid some tests on inputs which reduces significantly the number of simulations. The presented approach can in some case fail because of its dependency to many parameters that the user has to calibrate.

As a future work, a procedure to dynamically choose the parameters such as prediction period or additional constraints on system behavior could be considered, w.r.t. a prediction of the seabed for example.

References

[1] Julien Alexandre dit Sandretto and Alexandre Chapoutot. Validated Explicit and Implicit Runge-Kutta Methods. *Reliable Computing*, 22, 2016.

[2] Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Olivier Mullier. Formal verification of robotic behaviors in presence of bounded uncertainties. In *International Conference on Robotic Computing*. IEEE, 2017.

[3] A. Bemporad, A. De Luca, and G. Oriolo. Local incremental planning for a car-like robot navigating among obstacles. In *International Conference on Robotics and Automation*, pages 1205–1211, 1996.

[4] A. Censi, D. Calisi, A. De Luca, and G. Oriolo. A bayesian framework for optimal motion planning with uncertainty. In *International Conference on Robotics and Automation*. IEEE, 2008.

[5] E. R. Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker Inc.

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

[7] Luc Jaulin. *Mobile Robotics*. ISTE Press - Elsevier, 2015.

[8] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis*. Springer, 2001.

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

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

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

[12] José M. Mendes Filho and Eric Lucet. Multi-robot motion planning: a modified receding horizon approach for reaching goal states. *Acta Polytechnica*, 56(1):10–17, 2016.

[13] Ramon E. Moore. *Interval Analysis*. Prentice Hall, 1966.

[14] Nedialko S. Nedialkov, K. Jackson, and Georges Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. and Comp.*, 105(1):21 – 68, 1999.

[15] L. A. Page and A. C. Sanderson. Robot motion planning for sensor-based control with uncertainties. In *International Conferenc on Robotics and Automation*, volume 2, pages 1333–1340. IEEE, 1995.

[16] Romain Pepy, Michel Kieffer, and Eric Walter. Reliable robust path planning with application to mobile robots. *International Journal of Applied Mathematics and Computer Science*, 19(3):413–424, 2009.

[17] Michel Rueher. Solving continuous constraint systems. In *International Conference on Computer Graphics and Artificial Intelligence*, 2005.

[18] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 57(11):2817–2830, Nov 2012.