Reliable motion planner evaluated on a mobile robot

Elise Crépon, Adina M Panchea, Alexandre Chapoutot

Reliable motion planner evaluated on a mobile robot

Pierre-André Crépon
ENSTA ParisTech,
Palaiseau, France
crepon.pierre.andre@gmail.com

Adina M. Panchea
LIX, École Polytechnique,
Palaiseau, France
panchea@lix.polytechnique.fr

Alexandre Chapoutot
U2IS, ENSTA ParisTech,
Palaiseau, France
alexandre.chapoutot@ensta-paristech.fr

***Abstract*—Autonomous mobile robots need to be equipped with appropriate planification and control navigation systems in order to obtain safe behaviours. This study aims at implementing on a two wheeled mobile robot a robust autonomous navigation planning algorithm, which guarantees a safe and reliable path. First, making use of all the facilities that robot operating systems (ROS) middleware and the open motion planning library (OMPL) can offer an autonomous architecture is implemented on a mobile robot, with planning and control navigation systems adapted to the investigated problem. The planning navigation system make use of the widely-used Rapidly-exploring Random Tree (RRT) algorithm, while the control navigation level is based the go-to-goal strategy. As a novelty, a reliable and safe navigation planning algorithm based on RRT principles, *e.g.*, BoxRRT, and solved in an interval analysis framework, proposed in a one of our recent study is tested on a mobile robot platform. Through experiments, we demonstrate the utility of using such robust navigation planner in an autonomous navigation architecture, where uncertain localization is considered.**

I. Introduction

Autonomous navigation of mobile robots has attracted considerable attention of researchers in the areas of robotics and autonomous systems in the past decades [20]. One of the interests in the field of the autonomous navigation of mobile robot is the development of mobile platforms that robustly operate in complex or populated environments and offer various services without human interactions. For these reasons, the autonomous navigation problematic remains quite a challenge and it supposed to have navigation planning and control algorithms that will make mobile robots successfully accomplish a given mission while avoiding stationary or/and moving obstacles. Also, robust real-time visual tracking and objects detection algorithms may be considered for localization purposes.

Related work

Many planning algorithms have been proposed in the literature. In this study, more attention is given to approaches such as stochastic sampling which discretise the configuration space. More precisely, we focus on the Rapidly-exploring Random Trees (RRT) ([7], [9], [5], [8]) navigation planning algorithm, which covers the whole configuration space and easily integrates complex robot models.

A challenge for the navigation planners is related to the guarantee of the system's safety. This means that while planning, uncertainties usually resulting from: approximate localization, imperfect embedded sensors or approximate models used to describe the behaviour of the mobile robot devices should be taken into account. Considering uncertainties in the navigation planning level has been considered using several representations such as set-membership ([13], [15], [17]) or covariance matrices ([6], [16], [3]). 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 ([6], [16]) can guarantee the safety of the path at a certain confidence threshold.

Recently, some studies provided the guarantee of a safe path to imperfect proprioceptive sensors and/or localization information, while considering the uncertainties bounded with know bounds ([14], [17]). Under the latter uncertainty representation, [15] and [17] introduced reliable and robust navigation planners algorithms based on RRT principles and solved using interval analysis tools ([10], [4]).

The navigation control algorithms are also important in the autonomous navigation and must be developed to ensure that the mobile robots achieves the given mission by going towards the waypoints (ex. go to goal strategy) or following the path (pure pursuit algorithm) provided by the navigation planning level with any risk of collision with obstacles. Moreover, an interesting survey on some navigation control algorithms is proposed in [20].

Contributions

The main focus of this research study is to apply an autonomous navigation architecture, as reported in Fig. 1, on an easy to reproduce robotic platform with static obstacles and composed of a navigation planning level (to ensure a path with given initial and final configurations), a navigation control level (to achieve the desired goal configuration) and visual localisation provided by a low-cost camera. The navigation planning problem, as addressed in this paper, consists in finding waypoints, trajectory or a sequence of control policy to drive a mobile robot from a given initial to a given goal configuration while avoiding collision with given sets of obstacles. For the navigation planning level will make use, firstly, of a classical and well known planning algorithm (RRT) provided by the the open motion planning library (OMPL [19]). Secondly, a planning algorithm which can guarantee safe paths in an uncertain configuration space, where all

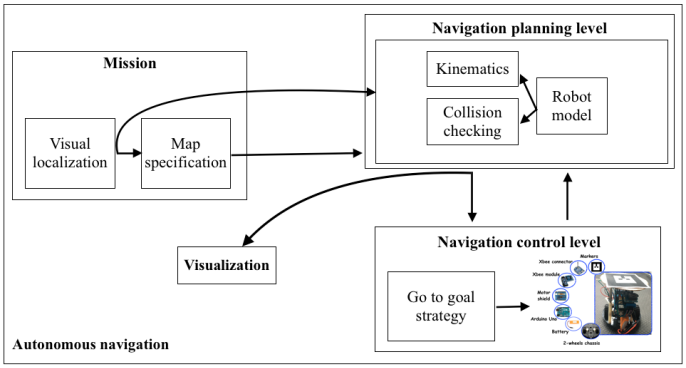


Figure 1. Autonomous navigation architecture

approximate initial and final mobile robot localisation are bounded with known bounds, *e.g.*, BoxRRT [17], will be considered. As in this paper the main focus is at the navigation planning algorithm, as stated previously, the navigation control level, *i.e*, the low-level one of the autonomous navigation architecture, will be based on PID controller along with the go-to-goal strategy, which can easily be formulated or found in the literature [18]. Finally the robotic platform is composed of low-cost hardware components, while the entire autonomous navigation achitecture is implemented using robot operating systems (ROS), C++ (through roscpp dependency) and the interval analysis library *e.g.* DynIBEX [2], used for the first time in an autonomous navigation architecture. Moreover, the DynIBEX library provides operators to deal with constraint satisfaction problems embedding differential equations. Briefly, the main contribution of this paper is twofold: *(i)* first, the safe and robust navigation planning algorithm *e.g* BoxRRT is formulated to underline and understand its practical usage in our defined framework and *(ii)* second, is implemented on a easy to reproduce mobile robot application by making use of ROS along with the DynIBEX library, while visual localisation information and round-off errors are bounded with known bounds.

II. Problem Statement

Context Given a mission, which consists in sending initial and goal configurations provided by a visual sensor (camera) to the navigation planning level, waypoints are then sent to the controller level so that a mobile robot to achieve while avoiding static obstacles.

Environment The mobile robot has to be driven in a two-dimensional static environment from an initial state to a desired one while avoiding obstacles, with known positions, represented by polygons shapes.

Problem Given unique initial and goal positions, find collision-free paths and drive the mobile robot from the initial to the goal position.

Solution First a two-wheeled mobile robot localised by a visual sensor is created to receive control informations via radio transmission, sent by the navigation planning level. The control informations consists in:

- waypoints provided by the OMPL - RRT algorithm adjusted to our case and
- sequence of control policy or waypoints given by the robust navigation planner, namely BoxRRT

which drives the mobile robot to the desired configuration, while making use of the architecture illustrates on Fig. 1.

Moreover, the used middleware witch connects the levels proposed in the autonomous navigation architecture from Fig. 1 is ROS.

III. Problem Formulation

The configuration space $\mathbb{S} = \mathbb{S}_{\text{free}} \cup \mathbb{S}_{\text{obs}}$ is 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.

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 initial condition $\mathbf{s}_0$ is denoted by $\mathbf{s}(t; \mathbf{s}_0)$.

A. Navigation planner

From an initial state $\mathbf{s}_0$ which belongs to the free configuration space $\mathbf{s}_0 \in \mathbb{S}_{\text{free}}$ the system needs to reach a given goal states $\mathbf{s}_{\text{goal}} \subset \mathbb{S}_{\text{free}}$ while avoiding the obstacle configuration space:

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

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

B. Robust navigation planner

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 purpose of the *robust navigation planner* is to provide a sequence of control inputs $\mathbf{u} \in \mathbb{U}^{\Delta t}_{[\mathbf{u}]}$ 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 $\mathbf{u} \in \mathbb{U}^{\Delta t}_{[\mathbf{u}]}$ 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 navigation planner for which there exists a sequence of control input $\mathbf{u} \in \mathbb{U}^{\Delta t}_{[\mathbf{u}]}$ to drive the system from an uncertain initial state to a set of goal states $\mathbb{S}_{\text{goal}}$. The formulation comes from [15] and is as follows:

$$\begin{array}{l} \exists K > 0 \text{ and } \mathbf{u} \in \mathbb{U} \text{ such that} \\ \forall [\mathbf{s}_0] \in \mathbb{S}_{\text{init}}, \forall [\mathbf{s}](K\Delta t; \mathbf{s}_0) \in \mathbb{S}_{\text{goal}} \text{ and} \\ \forall t \in [0, K\Delta t], [\mathbf{s}](t; \mathbf{s}_0) \in \mathbb{S}_{\text{free}}, \end{array} \tag{3}$$

with $[\mathbf{s}](t)$ the solution of (1). If a solution for (3) exists then the reliability of the robust path's reliability will be guaranteed.

Moreover, uncertainties related to its initial and final positions and orientation w.r.t. a frame attached to the environment are considered.

C. Interval analysis and Validated numerical integration

This section recalls notations deployed in this study regarding *interval vectors* or *boxes* ([4]) which are being used to represent the environment uncertainties.

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 $\text{Hull}(\mathbb{A})$. The inner approximation of a set $\mathbb{A}$, denoted $\text{Int}(\mathbb{A})$, is a box included in $\mathbb{A}$, *i.e.*, $\text{Int}(\mathbb{A}) \subset \mathbb{A}$. The Hausdorff distance [12] of two intervals $[x_1]$ and $[x_2]$ is

$$d([x_1],[x_2]) = \sup\{|\underline{x}_1 - \underline{x}_2|, |\overline{x}_1 - \overline{x}_2|\}. \tag{4}$$

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 $\mathbf{s}_0$, …, $\mathbf{s}_{\text{end}}$, 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 [11] and the references therein for more details.

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

$$R : \begin{cases} \mathbb{R} \mapsto \mathbb{IR}^n \\ t \to [\mathbf{s}] \end{cases} \tag{5}$$

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

$$\tilde{R} : \begin{cases} \mathbb{IR} \mapsto \mathbb{IR}^n \\ [\underline{t}, \overline{t}] \to [\tilde{\mathbf{s}}] \end{cases} \tag{6}$$

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

IV. Navigation planning algorithms

Let's start by formulating and briefly describing both navigation algorithms.

***Global description*:**

First, the navigation planner samples a random state in the state space. Then finds the state among the ones in the tree of states which is closest to it. Next, the latter found closest state expands towards the random one, until a new state is reached. If the new state is reached without collision it will be added to the exploration tree of states, as suggested by Algorithm 1.

input : $\{\text{state}_{\text{init}}, \text{state}_{\text{goal}}\} \subset \mathbb{S}_{\text{free}}, \Delta t \in \mathbb{R}^+, \text{K} \in \mathbb{N}$
output: G

1 G.init($\text{state}_{\text{init}}$);
2 i $\leftarrow 0$;
3 **repeat**
4 $\quad \text{state}_{\text{rand}} \leftarrow$ random(i);
5 $\quad$ G$\leftarrow$EXTEND(G, $\text{state}_{\text{rand}}$, $\text{state}_{\text{goal}}$);
6 **until** i++ $<$ *MaxIter*;
7 **return** G or Failure if no such path exists;

Algorithm 1: RRT and BoxRRT planner algorithm

Next, each of the two planning algorithms are introduced.

A. OMPL - RRT navigation planner algorithm : brief recall

The first algorithm used in the navigation planner level is an adapted version of the geometric planner algorithm RRT, which accounts for geometric constraints of a system. Moreover, the interface and the structure for the path planning is provided by OMPL, which was adapted with further functionalities for path analysis and visualization.

***Description*:**

First the given initial configuration $\mathbf{s}_{\text{init}}$ is added to the exploration tree G (Algo. 1 Line 1). Then, a random state $\mathbf{s}_{\text{rand}} \subset \mathbb{S}_{\text{free}}$ is chosen by the procedure *random* (Algo. 1 Line 4). The *nearest-neighbor* procedure from Procedure 2 Line 1 returns the closest vertex $\mathbf{s}_{\text{near}}$ to $\mathbf{s}_{\text{rand}}$ in the tree G, according to a certain metric. The closest vertex $\mathbf{s}_{\text{near}}$ expands towards the random one, until a new state $\mathbf{s}_{\text{new}}$ is reached (Procedure 2 Line 2). If it can be proved that the edge (typically a straight lines) between $\mathbf{s}_{\text{near}}$ and $\mathbf{s}_{\text{new}}$ is a *collision free path*, then $\mathbf{s}_{\text{new}}$ is added to G as a new vertex *G.add-vertex* procedure and connected to its parent $\mathbf{s}_{\text{near}}$ though the *G.add-edge* procedure.

1 $\mathbf{s}_{\text{near}} \leftarrow$ nearest-neighbor($G, \mathbf{s}_{\text{rand}}$);
2 $\mathbf{s}_{\text{new}} \leftarrow$ new-state($\mathbf{s}_{\text{rand}}, \mathbf{s}_{\text{near}}, \mathbf{s}_{\text{goal}}$);
3 **if** *collision-free-path* ($\mathbf{s}_{near}, \mathbf{s}_{new}$) **then**
4 $\quad$ G.add-vertex($\mathbf{s}_{\text{new}}$);
5 $\quad$ G.add-edge($\mathbf{s}_{\text{near}}$, $\mathbf{s}_{\text{new}}$);
6 **return** $\emptyset$;
7 **return** G

Algorithm 2: EXTEND$_{\text{RRT}}$ procedure

B. BoxRRT navigation planner algorithm

Several versions of BoxRRT can be found in [15], [14], [17] for the case where the uncertainties related to the configuration space are considered on the final and the initial states. In this paper, we have interest in using as the navigation planner level the version of BoxRRT proposed in one of our recent study [17], which also presents the differences between the robust motion planners. In the followings the planner algorithm and the used procedures are introduced.

***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 as in (4). 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 K 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.

1 $[\mathbf{s}_{\text{near}}] \leftarrow$ nearest-box-neighbor(G, $[\mathbf{s}_{\text{rand}}]$);
2 $[\mathbf{s}_{\text{new}}] \leftarrow$ new-box-state($[\mathbf{s}_{\text{rand}}], [\mathbf{s}_{\text{near}}], \mathbf{u}, \Delta t, [\mathbf{s}_{\text{goal}}]$);
3 **if** *collision-free-path* ($[\mathbf{s}_{near}], [\mathbf{s}_{new}], \mathbf{u}, \Delta t$) **then**
4 | G.add-guaranteed-vertex($[\mathbf{s}_{\text{new}}]$);
5 | G.add-guaranteed-edge($[\mathbf{s}_{\text{near}}]$, $[\mathbf{s}_{\text{new}}]$, $\mathbf{u}$, Δt);
6 **return** $\emptyset$;
7 **return** G

Algorithm 3: EXTEND$_{\text{BoxRRT}}$ procedure

***Random procedure*:** This procedure, previously proposed in [15], consists in choosing the random state in the final configuration state $[\mathbf{s}_{\text{rand}}] \subset [\mathbf{s}_{\text{goal}}]$ with a probability $p > 0$ which is also known as the *Random GoalBias procedure*.

***Nearest box neighbor procedure*:** Finds the closest vertex to the $[\mathbf{s}_{\text{rand}}]$ one according to a chosen metric d, here the Hausdorff distance between two intervals as defined in (4).

***New box state 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 III-C 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 [1], which are capable of testing during the integration procedure if a collision occurred. Therefore this procedure differs from the previously BoxRRT proposed in [15], [14] which uses wrap techniques. 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, being more elegant and improving the wrapping effect produced by the wrap techniques.

V. Results

This section presents the results obtained after implementing the autonomous navigation achitecture, reported on Fig. 1, the two types of navigation planners described in Sect. IV. The autonomous architecture corresponds to the experimentation protocol, presented in Sect. V-A for which the configuration space size is 1.2m $\times$ 1.8m $\times$ 2πrad.

Three initial configurations are considered (illustrated on fig. 2), on the same map, with the same goal position (0.72m;0.34m;$-$0.04rad). The initial configurations change in such a way that the difficulty and distance between the initial and goal configurations increase from one configuration to another, such as: *Config. (1)* has no obstacles between the mobile robot and the goal configuration. For this case the initial position size is (0.71m;$-$0.46m;$-$0.003rad). *Config. (2)* has static obstacles between the mobile robot and the goal configuration, where the initial position size is (0.03m;0.01m;$-$0.01rad). *Config. (3)* corresponds to a labyrinth map with static obstacles, in which the initial position size is ($-$0.77m;$-$0.44m;$-$0.004rad).

The initial and goal configurations boxes, for the BoxRRT planner case, are build by choosing as boxes middles the values states previously (the ones used for the RRT planner) to which we add: $\pm$0.02cm of error on both x, y position and $\pm$0.01 rad for the orientation for the initial case and $\pm$0.05 cm of error on both x, y position and the orientation being allowed between $[-\pi/2; \pi/2]$.

All simulations and experimental tests were performed on an Intel Core m7-6Y75 CPU at 1.20GHz$\times$4. We recall that the used library for implementing the guaranteed BoxRRT planner, as stated in the introduction section, consists in DynIBEX.

A. Platform architecture and description

The mechanical design of the mobile robot was kept as simple as possible, which can be spotted on Fig. 1 in the navigation control level. As the base of the physical system, we used a two-wheeled chassis, motor drivers with encoders, a ball caster as a passive wheel element and the following

devices: power supply (7V battery), Arduino uno and the motor shield, communication (radio transmission at 2.4 Gz based ZigBee via Xbee modules) and localization marker (necessary for the AR tag tracking library used with ROS). Motors receive the information via the Xbee modules and the rosserial ROS package makes sure that the connection between the communication and Arduino is correctly made. The planners send a list of waypoints to the navigation control level, which utilises the go-to-goal strategy and ensures the mobile robot achieves the given goal location. The robot's two motors allow for simple longitudinal speed and steering control. This project was written in ROS environment and a brief description of the experimental architecture is as follows: First, a map file (ppm format), in which the free and the non-admissible configuration, respectively are specified with colour codes: whiter for the admissible places and darker for the non-admissible ones. A ROS AR tag tracking package, which requires the use of a camera and specific QR codes, is used to specify the configurations at which the mobile robot will begin and end its path. Then, the latter information is supplied to the planner, which generates waypoints for a collision-free path which the mobile robot needs to achieve. The map, the waypoints and the QR codes can be visualized through the camera, and one can easily use the Rviz visualiser. Finally, the waypoints are sent to the Arduino board via radio communication server based Xbee modules which will make the mobile robot to go towards the desired configuration.

B. Robot mobile modeling

The differential system which describes the evolution of the mobile robot is represented by the kinematic model of an unicycle robot type:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega, \tag{7}$$

with (x,y) the position and θ the orientation w.r.t. a frame attached to the environment. The control input $u = [v\omega]$ is represented by the linear $v \in [-0.4; 0.4]$ m/s and the angular $\omega \in [-10; 10]$ rad/s velocities respectively. The latter values were determined experimentally.

C. Performance analysis

Each navigation planner used for the experimental protocol is first performed 100 times, in simulation, in order to verify the computational time. For the geometric RRT navigation planner, which is widely used and provided by the OMPL, a benchmark [21] can be performed locally or online on plannerarena.org. The obtained CPU obtained was less than 1 s for all of the performed 100 simulations. On the other hand, the BoxRRT navigation planner algorithm being at the beginning of its use, and not integrated in an online library (which will be part of a future work) a CPU analysis is required. The robust navigation planner increases the computational time while the desired configurations are changing as follows: 21s for Config. 1, 38s for Config. 2 and 65 s for Config. 3. The reported CPU values are the mean of the CPU obtained after simulating it 100 times. It is not surprising to obtain a larger computational time for the BoxRRT algorithm as it always requires numerical guaranteed integration of the system, which is not the case for the RRT one which here is easily formulated as a geometric navigation planner only based on geometric constraints. Still, a benchmark between these two navigation planners cannot be done due to the differences among them, as stated previously one is based on geometrical metrics and the other utilises interval analysis to provide a numerical guaranteed integration. This study is not focusing on comparing the two navigation algorithms, the purpose being to introduce the BoxRRT navigation algorithm for real application and to presents the advantages of this planner usage, which was never reported before.

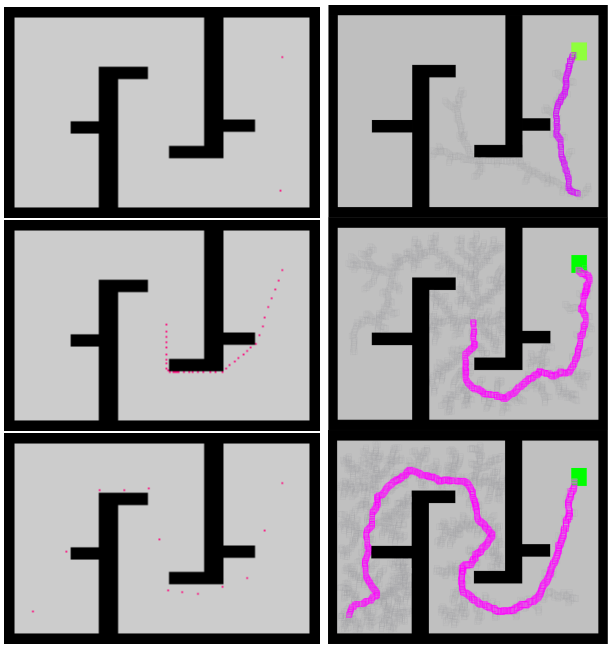
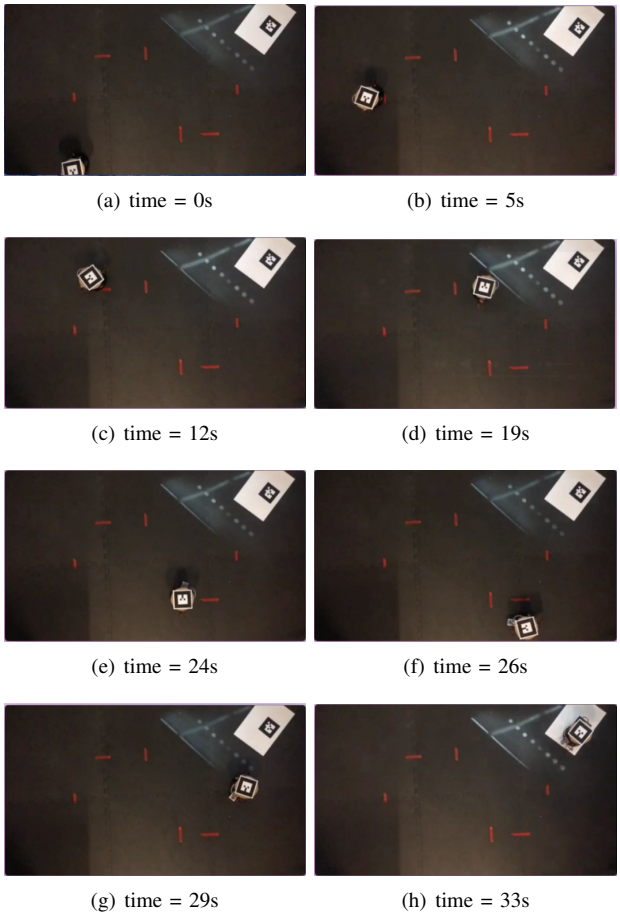


Figure 2. Planned waypoints (on the left side) by the RRT planner and planned trajectory tube (on the right side) by the BoxRRT planner.

Fig. 2 presents the waypoints planned by the RRT planner via OMPL and the trajectory tube obtained by the BoxRRT one. For the RRT case the waypoints information are sent to the controller level, for which a go to goal strategy is applied and moves the robot towards the goal why passing through the waypoints provided by the planner. In the case of the BoxRRT we decide to send the middle of the obtained boxes along the path, but any other position belonging to the boxes of waypoints can be selected or one can provide trajectory for the exhibited tube of trajectories. Once the middles are provided, the controller level will act as for the RRT planner algorithm case. In both navigation planner cases, the controller level sends motor commands which moves the mobile robot from the initial configuration to the desired one. One illustrative example of the mobile robot's behaviour with the commands received by the controller level is reported on Fig. 3 for the third configuration.

Both navigation planners found waypoints for free-collision paths. Even so, the mobile robot localization provided by



(a) time = 0s (b) time = 5s

(c) time = 12s (d) time = 19s

(e) time = 24s (f) time = 26s

(g) time = 29s (h) time = 33s

Figure 3. An illustration of the mobile robot going towards and achieving the labyrinth configuration.

visual means and which depends on QR code can be perturbed due to some external perturbations. In the latter hypothesis and by using the RRT planner the user need to physically place the mobile robot at the precise desired location. While for the BoxRRT planner an imprecise initial position can be fixed by defining an initial region and a tube is provided which takes into consideration the uncertain locations. Moreover, as long as the robot stays inside the provided boxes of waypoints or tube of trajectories it is guaranteed that the robot achieves the task with no collision.

VI. Conclusions and future works

Though this study an implementation of an autonomous navigation schema is presents on a two-wheeled mobile robot. The navigation level is composed of planners based sampling-based planners, namely RRT. The proposed planners for the autonomous schema are already reported in the literature, while one of them, the RRT, being widely used. Even so, the second one which considers uncertainties on initial and final configurations belonging to boxes, *i.e.*, the BoxRRT, is used and implemented on a real mobile robot application for the first time.

In future work we plan to provide the BoxRRT by attaching it to an online library. Also some growth of the boxes can occur due to numerical round-off and errors and due to the over-approximation used in the guaranteed integration which may increase the imprecision at each new uncertain configuration. For this reason in a future version of BoxRRT planner the localization information will be update from time to time and the planner will piecewise plan new collision free path along the initially plan path.

Acknowledgment

This work was supported by DGA MRIS. Pierre-André Crépon was a student at Lycée Hoche, Versailles, France during his internship at ENSTA ParisTech.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] J. J. 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.

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

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

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

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

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

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] A. M. Panchea, A. Chapoutot and D. Filliat. Extended Reliable Robust Motion Planners. In *International Conference on Decision and Control*, pp 1112–1117, 2017.

[18] M. Egerstedt. Control of mobile robots. 2013.

[19] I.A. Şucan, M. Moll and L.E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 18(4):72–82, 2012.

[20] B. Paden, M. Cap, S. Z. Yong, D. Yershov, E. Frazzoli A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[21] M. Moll, I.A. Şucan and L.E. Kavraki. Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization. *IEEE Robotics & Automation Magazine*, 2015.