Complexity Study of Guaranteed State Estimation for Real Time Robot Localization

Submitted 19th July 2010; accepted 10th December 2010.

Emmanuel Seignez, Alain Lambert

Abstract:

The estimation of a vehicle configuration in its environment is mostly solved by Bayesian methods. Interval analysis allows an alternative approach: bounded-error localization. Such an approach provides a bounded set of configuration that is guaranteed to include the actual vehicle configuration. This paper describes the bounded-error localization algorithms and presents their complexity study. A real time implementation of the studied algorithms is validated through the use of an experimental platform.

Keywords: *autonomous robot, data fusion, localization, bounded error state estimation, algorithm.*

1. Introduction

The aim of the localization problem is to maintain a correct vehicle configuration (position and orientation) among its displacement using proprioceptive and exteroceptive sensors [3]. Proprioceptive sensors generate a cumulative error along the displacement. Consequently, using proprioceptive sensors does not give a satisfactory positioning to be used in higher level tasks like path following or path planning. To cope with this problem, any localization processes also use exteroceptive sensors to improve the predicted configurations of the vehicle. Thus, the localization processes are broken down into two steps: they alternate the prediction phases using proprioceptive sensors and a correction (estimation) phase using exteroceptive sensors.

The most commonly used methods are based on Kalman filtering [9], [19], [8] and Markov localization, using either a probability grid [5] or particle filtering [2], [29].

Kalman filtering [7] (in its basic implementation) requires neither a great amount of computing power nor memory but, in return, cannot perform global localization and can only track one configuration of the vehicle. Moreover, it diverges very swiftly in the presence of erroneous data (outliers) even whilst using methods developed to overcome some of its limitations such as those reported in [10]. Kalman filtering is therefore not well adapted to a situation where outliers are unavoidable.

Markov localization methods require more computing power than Kalman filtering but provide more reliable estimated configurations in complex, dynamic or badly mapped environments [6]. These methods have dominated for the last few years and much work is still going on to improve them, especially Monte Carlo localization [30]. Markov localization methods evaluate the probability of a vehicle being in a given region of a mapped environment but nothing ensures that the vehicle is indeed in the configuration with the highest probability.

Bounded-error state estimation [11] is an alternative and less known method which has been proposed for localization [21], [22] and tracking [13]. This method is based on R. E. Moore's [23] work on interval analysis. He proposed to represent a solution to a problem by giving an interval in which the real solution is guaranteed to be. Thus, whereas the majority of localization methods provide probabilistic results, this method gives a set of bounded configuration in which the vehicle is guaranteed to be.

In bounded-error state estimation, all model and measurement errors are assumed to be bounded, with known bounds. At each time instant, the bounded-error recursive state estimator provides a set containing all possible configurations of the vehicle given the measurements and noise bounds. The methodology has proved its feasibility in simulations [13]. Experiments were done to demonstrate that this method can be made operational on real data [25], [26]. [26] suggests using the volume of bounded localization in order to tune the method with a number of boxes that remains tractable. Nevertheless, [26] does not show how to achieve it. This paper focuses on the bounded localization algorithms of the method and shows how to tune it in order to achieve a real time operation. Whereas [12], [13] focus on the mathematical aspects of the method, a complete presentation of the algorithms is provided here. These algorithms are then used to calculate the complexity of the localization method and to prove its tractability. Finally, we will propose a new method of automatically setting parameters to provide the most precise localization in real time.

Section 2 describes the necessary mathematical tools based on interval analysis. Section 3 shows the way of representing the solution set with subpavings. Section 4 presents the localization process: the operations realized during the prediction step are reported in Section 4.1 whilst Section 4.2 presents the estimation step. Section 5 shows the complexity of each studied algorithm and Section 6 explains how to tune the parameter so as to achieve a real time implementation. Finally, Section 7 presents the experimental platform and an experimental validation of the proposed real time bounded error state estimation.

2. Interval analysis

2.1. Overview

Bounded-error state estimation is based on interval analysis. Interval analysis was introduced in the sixties in order to avoid the problem of approximations in calculations. R. E. Moore [23] proposes to represent a solution of a problem by an interval in which the real solution is guaranteed to be. The minimum and maximum of this interval are perfectly known and not approached. Interval analysis provides a set of rules to calculate with the interval $[x] = [\underline{x},\overline{x}] \subset \mathbb{R}$ where $\underline{x}$ and $\overline{x}$ are respectively the minimum and the maximum of $[x]$. The width of an interval is $w[x] = \overline{x} - \underline{x}$. Arithmetical operations (+, –, * and /) and standard mathematical functions readily extend to intervals. For example,

$$[1,2]+[3,4]=[4,6]$$
$$\ln([1,e])=[0,1] \qquad (1)$$

2.2. A free library

The computing development using the interval analysis is simplified by the use of PROFIL/BIAS (Programmer's Runtime Optimized Fast Interval Library/Basic Interval Arithmetic Subroutines) [16], [15], [1], a C++ class library supporting the most commonly needed interval and real operations in a user friendly way. Using this library allows the manipulation of intervals as numbers. All basic mathematical functions were implemented to accept numbers as well as intervals.

2.3. Inclusion function

The notion of *inclusion function* is one of the most important tools provided by interval analysis [11]. For any function where $f : D \subset \mathbb{R} \rightarrow \mathbb{R}$ is defined as combinations of arithmetical operators and elementary functions, interval analysis makes it possible to build inclusion functions f satisfying

$$\forall [x] \subset D, f([x]) \subset f_{[\,]}([x]) \qquad (2)$$

where $f([x])$ denotes the set of all values taken by $f(.)$ over $[x]$.

The simplest way to obtain an inclusion function is to replace all real variables by interval ones and all real-valued operators or elementary functions by their interval counterparts. The *natural inclusion function* is then obtained. For example, the function shown below

$$f(x)=x^2-x+1 \qquad (3)$$

has the following natural inclusion function

$$f_{[\,]}([x])=[x]^2-[x]+1 \qquad (4)$$

For *convergent* functions (all functions considered in this work are convergent), the width of their image interval tends to zero when the width of the corresponding argument interval tends to zero. As a consequence, cutting the interval into smaller intervals improves the result of the inclusion function (see [24], [11]). For instance using Eq. (3), $f_{[\,]}([0,2])$ gives a result of $[-1,5]$, whereas a calculation of $f_{[\,]}([0,1]) \cup f_{[\,]}([10,2])$ gives an interval of $[0,4]$ which is better than $f_{[\,]}([0,2])$.

Outer-approximation of sets may be achieved by a union of non-overlapping boxes or *subpaving*. Subpaving combined with direct image evaluation and inverse image evaluation algorithms are the building stones of the bounded-error state estimation algorithm.

3. Dealing with subpavings

3.1. Introduction

The vehicle configuration in the global frame is denoted by $\mathbf{x} = (x,y,\theta)^T$ where (x,y) are the coordinates of the middle of the segment joining the two steering wheels and θ specifies the orientation of a local frame attached to the vehicle with respect to the global frame.

To estimate and handle the sets implied in our problems, we use the concept of boxes as bounded configurations: a box $[\mathbf{x}] \in \mathbb{R}^3$ is composed of 3 intervals $[\mathbf{x}]=[x]\times[y]\times[\theta]$. A subpaving [11] is a union of non-overlapping boxes. In the following sections, a description of the operations performed to calculate the subpaving is provided.

3.2. Use of a subpaving in localization process

The principle of localization consists of testing the binary relevance of each box according to the data returned by the sensors. Firstly we verify if a box (or a part of it) is compatible with the measurement: if the answer is positive the box is kept, else the box is discarded. If only a part of the box is compatible with the measurement, then the box is cut into smaller boxes making it possible to improve the solution description (see Section 2.3). In order to make the operations more efficient, two different models are used to order the description and the storage of these boxes (see Section 4). The set of vehicle configurations is either represented as a list in which overlapping boxes are present, or as a binary tree which avoids overlapping boxes. In the tree representation, each node has the description of a box and the box located at the root of the tree contains all the boxes in the tree.

3.3. Tree organization

The binary tree is constructed by dichotomy: a box is cut into two children boxes by interval bisection (one dimension of the box is bisected). These boxes have the same subintervals length in one dimension and a copy of all the other dimensions. We chose to cut the box according to the largest length. For instance, cutting the root box $[\mathbf{x}]$ among y dimension lead to two boxes named $L[\mathbf{x}]$ and $R[\mathbf{x}]$ (corresponding respectively to the right and the left children $[\mathbf{x}]$):

$$L[\mathbf{x}] = \left[\underline{x},\overline{x}\right]\times\left[\underline{y},\frac{\underline{y}+\overline{y}}{2}\right]\times\left[\underline{\theta},\overline{\theta}\right],$$
$$R[\mathbf{x}] = \left[\underline{x},\overline{x}\right]\times\left[\frac{\underline{y}+\overline{y}}{2},\overline{y}\right]\times\left[\underline{\theta},\overline{\theta}\right]. \qquad (5)$$

3.4. Representation of a set using a subpaving

The Figure 1 gives a two dimensions example of a subpaving calculation for a set $\Omega \in \mathbb{R}^2$ bounded on $[0;5]\times[0;5]$.

Starting from a root $[0;5]\times[0;5]$ the extrema of Ω are calculated to reduce the box. Here we find $[0.1;4]\times[0;3.8]$ (Figure 1.1). The bisection takes place using the largest length dimension. The two studied boxes are $[0.1;2.05]\times[0;3.8]$ and $[2.05;4]\times[0;3.8]$ (Figure 1.2). In order to reduce the boxe's length, we calculate the extrema of Ω for each box. On the right child, the box is reduced to $[2.05;4]$

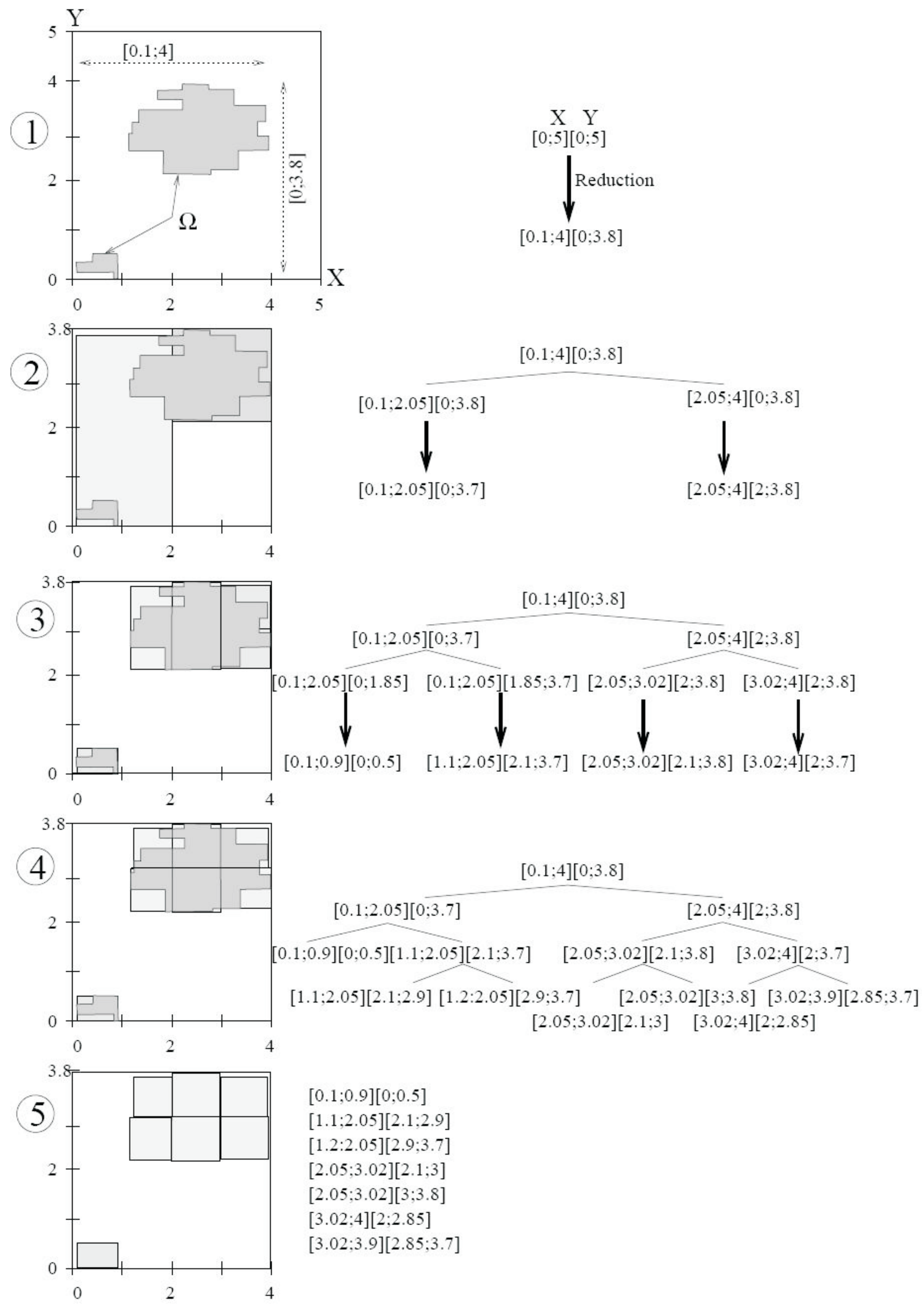


Fig. 1. Binary tree and list representation of a set of configurations.

×[2;3.8] because the interval [2.05;4]×[0;2] is empty. Similar operations are done for the left child which is reduced to [0.1;2.05]×[0;3.7]. Each box is then bisected again. This bisection is done until it meets one of the three following criteria:

1 - a box is completely inside Ω, in this case the box is kept,
2 - a box is not contained by Ω, in this case the box is deleted,
3 - the bisected box reaches the minimum length, in this case the box is kept.

3.5. Converting a tree into a list

The actual configuration of the vehicle is included into a leaf node. For prediction steps (in the localization process), a complex tree organization of the subpavings is not needed. When only the leaf nodes are required to describe the subpavings, the boxes can be organized as a list of nodes. Doing that reduces the number of nodes and thus the quantity of memory and calculation needed. A function TREE_TO_LIST (Algorithm 1) is recursively called to visit the tree until a leaf node is found. This leaf node is placed in the list. The described algorithm has been used here to calculate the list of Figure 1.5 from the tree in Figure 1.4.

3.6. Converting a list into a tree (and eliminating overlapping boxes)

To reduce the number of leaf nodes (which increases during the prediction step) and thus to reduce the calculations, the overlapping boxes have to be deleted. For this purpose, the BUILDSP algorithm generates a set of non-overlapping boxes ordered as a binary tree. BUILDSP starts from possibly overlapping boxes ordered as a list.

The Algorithm 2 (BUILDSP), deduced from the explanation in [12], reduces the number of pavings and the future calculations. Algorithm 2 starts (in the first call) with a box $[\mathbf{x}]_{root}$ that contains all the box in the list "List". On line 2-9, $[\mathbf{x}]_{root}$ is reduced to eliminate all the areas that are not useful. This is necessary, on each recursive call, in order to bind the set as close as possible (see Figure 1.2). Line 10-12, Algorithm 2, BUILDSP tests if $[\mathbf{x}]_{root}$ is included in one box in the list. If yes, or $[\mathbf{x}]_{root}$ is smaller than ε (line 13), then the box is kept. If this is not the case $[\mathbf{x}]_{root}$ is cut among its larger dimension: it builds $[\mathbf{x}]_{child_{left}}$ and $[\mathbf{x}]_{child_{right}}$ (Line 14, Algorithm 2). The list of boxes is then divided into two sub-lists: List_{left} contains the boxes partially or totally in $[\mathbf{x}]_{child_{left}}$, List_{right} contains the boxes partially or totally in $[\mathbf{x}]_{child_{right}}$ (Line 15-24, Algorithm 2). The function BUILDSP is then called recursively for the two boxes $[\mathbf{x}]_{child_{left}}$ and $[\mathbf{x}]_{child_{right}}$ with their associated lists (Line 25-26, Algorithm 2). Finally, BUILDSP deletes the two sublists (right and left) and returns a tree.

The Algorithm 3 (called by Algorithm 2) calculates the box that includes a list of boxes. Starting from an empty box (line 1), the algorithm increases the bounding box according to the boxes in the list (lines 2 à 9).

Subpavings are used in a number of operations that are involved in the localization process.

4. Localization process

4.1. Prediction step

The prediction step uses the data provided by the odometers mounted on the rear wheels of our experimental platform (see Section 7 for a description of the sensors).

4.1.1. Odometric data integration

A non-linear discrete-time state-space model is considered to describe the evolution of the configuration $\mathbf{x}_k$ of the vehicle

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{v}_k) \tag{6}$$

where $\mathbf{u}_k$ is a known two-dimensional control vector which is assumed constant between the times indexed by k and k+1. $\mathbf{v}_k$ is an unknown state perturbation vector that accounts for the uncertain description of reality by the model.

X is a subpaving of $\mathbb{R}^3$ in which the robot is guaranteed to be. Integrating the odometric data is done by creating the subpaving that includes $Y = \mathbf{f}(X)$ where $\mathbf{f}$ is a known non-linear function. $X_{k|k-1}$, the set that is consistent with sensor data provided at time k knowing k−1, could be calculated using

$$\begin{aligned} X_{k|k-1} &= \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{v}) \text{ with} \left\{ \begin{array}{l} \mathbf{x} \in X_{k-1|k-1}, \\ \mathbf{u} + \mathbf{v} \in [\mathbf{u} + \mathbf{v}] \end{array} \right\} \\ &= \mathbf{f}(X_{k-1|k-1}, \mathbf{u} + \mathbf{v}). \end{aligned} \tag{7}$$

with $[\mathbf{u} + \mathbf{v}]$ the state vector $\mathbf{u}$ added to the bounded error $\mathbf{v}$ on the measurement. $[\mathbf{u} + \mathbf{v}]$ can be deduced from data sent by the incremental coder which are in binary format: the actual position of the coder for the right wheel belongs to

Algorithm 1 Convert a tree to a list

```
        list TREE_TO_LIST (tree Tree)
 1-   List ← ∅
 2-   if Tree.child_left ≠ ∅ then
 3-       List←List + TREE_TO_LIST (Tree.child_left)
 4-   end if
 5-   if Tree.child_right ≠ ∅ then
 6-       List←List + TREE_TO_LIST (Tree.child_right)
 7-   end if
 8-   if Tree.child_left= ∅ and Tree.child_right= ∅ then
 9-       List←List +Tree.root
10-   end if
11-   return List
```

Algorithm 2 BUILD a tree SubPaving from a list of boxes

```
tree BuildSp (paving $[\mathbf{x}]_{root}$, list List, float $\varepsilon$)
1-  if List= $\emptyset$ then return NULL end if
2-  $[\mathbf{x}]_{bound}$ $\leftarrow$BOUNDING(List)
3-  if $\underline{x}_{root} < \underline{x}_{bound}$ then $\underline{x}_{root} \leftarrow \underline{x}_{bound}$ end if
4-  if $\overline{x}_{root} > \overline{x}_{bound}$ then $\overline{x}_{root} \leftarrow \overline{x}_{bound}$ end if
5-  if $\underline{y}_{root} < \underline{y}_{bound}$ then $\underline{y}_{root} \leftarrow \underline{y}_{bound}$ end if
6-  if $\overline{y}_{root} > \overline{y}_{bound}$ then $\overline{y}_{root} \leftarrow \overline{y}_{bound}$ end if
7-  if $\underline{\theta}_{root} < \underline{\theta}_{bound}$ then $\underline{\theta}_{root} \leftarrow \underline{\theta}_{bound}$ end if
8-  if $\overline{\theta}_{root} > \overline{\theta}_{bound}$ then $\overline{\theta}_{root} \leftarrow \overline{\theta}_{bound}$ end if
9-  Tree.root$\leftarrow$ $[\mathbf{x}]_{\mathrm{root}}$
10- for each $[\mathbf{x}]_{\mathrm{list}} \in$ List do
11-     if $[\mathbf{x}]_{\mathrm{root}} \subset [\mathbf{x}]_{\mathrm{list}}$ return Tree
12- end for
13- if all length of $[\mathbf{x}]_{\mathrm{root}} < \varepsilon$ then return Tree end if
14- Cut $[\mathbf{x}]_{\mathrm{root}}$ into $[\mathbf{x}]_{\mathrm{child_{lelft}}}$, $[\mathbf{x}]_{\mathrm{child_{right}}}$ among the
                                         larger dimension
15- $\mathrm{List}_{\mathrm{right}} \leftarrow \emptyset$
16- $\mathrm{List}_{\mathrm{left}} \leftarrow \emptyset$
17- for each $[\mathbf{x}]_{\mathrm{list}} \in$ List do
18-     if $[\mathbf{x}]_{\mathrm{child_{left}}} \cap [\mathbf{x}]_{\mathrm{list}} \neq \emptyset$ then
19-         $\mathrm{List}_{\mathrm{left}} \leftarrow \mathrm{List}_{\mathrm{left}} + [\mathbf{x}]_{\mathrm{list}}$
20-     end if
21-     if $[\mathbf{x}]_{\mathrm{child_{right}}} \cap [\mathbf{x}]_{\mathrm{list}} \neq \emptyset$ then
22-         $\mathrm{List}_{\mathrm{right}} \leftarrow \mathrm{List}_{\mathrm{right}} + [\mathbf{x}]_{\mathrm{list}}$
23-     end if
24- end for
25- Tree.left_child $\leftarrow$ BuildSp($[\mathbf{x}]_{\mathrm{child_{left}}}$,$\mathrm{List}_{\mathrm{left}}$,$\varepsilon$)
26- Tree.right_child $\leftarrow$ BuildSp($[\mathbf{x}]_{\mathrm{child_{right}}}$,$\mathrm{List}_{\mathrm{right}}$,$\varepsilon$))
27- Delete ($\mathrm{List}_{\mathrm{right}}$)
28- Delete ($\mathrm{List}_{\mathrm{left}}$)
29- return Tree
```

Algorithm 3 Bound a list of boxes

```
box Bounding (list List)
1-  $[\mathbf{x}]_{bound} \leftarrow [+\infty, -\infty], [+\infty, -\infty], [+\infty, -\infty]$
2-  for each box $[\mathbf{x}]_{list}$ $\in$List do
3-      if $\underline{x}_{list} > \underline{x}_{bound}$ then $\underline{x}_{bound} \leftarrow \underline{x}_{list}$ end if
4-      if $\overline{x}_{list} < \overline{x}_{bound}$ then $\overline{x}_{bound} \leftarrow \overline{x}_{list}$ end if
5-      if $\underline{y}_{list} > \underline{y}_{bound}$ then $\underline{y}_{bound} \leftarrow \underline{y}_{list}$ end if
6-      if $\overline{y}_{list} < \overline{y}_{bound}$ then $\overline{y}_{bound} \leftarrow \overline{y}_{list}$ end if
7-      if $\underline{\theta}_{list} > \underline{\theta}_{bound}$ then $\underline{\theta}_{bound} \leftarrow \underline{\theta}_{list}$ end if
8-      if $\overline{\theta}_{list} < \overline{\theta}_{bound}$ then $\overline{\theta}_{bound} \leftarrow \overline{\theta}_{list}$ end if
9-  end for
10- return $[\mathbf{x}]_{bound}$
```

$[nt_r+1, nt_r+1]$ and $[nt_l+1, nt_l+1]$ for the left wheel. Knowing the coder resolution n_{coder}, the angular displacements of the left and right wheels are given by

$$[\delta r] = 2 \times [\pi] \times \frac{[nt_r-1, nt_r+1]}{n_{coder}}$$

$$[\delta l] = 2 \times [\pi] \times \frac{[nt_l-1, nt_l+1]}{n_{coder}} \qquad (8)$$

As π cannot be represented exactly, it is bounded. We can deduce the longitudinal motion δs and rotational motion $\delta\theta$ by integrating the measurement errors on the wheels radius R and on the distance between the two rear wheels $2e$:

$$[\delta s] = [R - \Delta R, R + \Delta R] \times \frac{[\delta l]+[\delta r]}{2} \qquad (9)$$

$$[\delta\theta] = [R - \Delta R, R + \Delta R] \times \frac{[\delta l]-[\delta r]}{2[e - \Delta e, e + \Delta e]} \qquad (10)$$

where R is the measurement error on the wheel radius and e the measurement error on the distance between the two rear wheels. These parameters are made larger than usual to allow for the wheel slipping.

The classical evolution model, described in [28], [17], is considered:

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} x_k + \delta s.\cos\left(\theta_k + \frac{\delta\theta}{2}\right) \\ y_k + \delta s.\sin\left(\theta_k + \frac{\delta\theta}{2}\right) \\ \theta_k + \delta\theta \end{pmatrix}, \quad (11)$$

Applying bounded parameters on (12) for each box $[\mathbf{x}] \in \hat{X}_{k-1|k-1}$ the bounded evolution of the vehicle is described by the following inclusion function:

$$\mathbf{f}_{[\,]}([\mathbf{x}]) = \begin{pmatrix} [x_k] + [\delta s].\cos\left([\theta_k] + \frac{[\delta\theta]}{2}\right) \\ [y_k] + [\delta s].\sin\left([\theta_k] + \frac{[\delta\theta]}{2}\right) \\ [\theta_k] + [\delta\theta] \end{pmatrix}, \quad (12)$$

An outer-approximation $X_{k|k-1}$ may be calculated using the IMAGESP (IMAGE SubPaving) algorithm [11], because the set of boxes $X_{k-1|k-1}$ is described by a subpaving and an inclusion function for the prediction function **f** is available.

4.1.2. IMAGESP

The IMAGESP algorithm (Algorithm 5) may be decomposed into three steps:

Mincing the subpavings (line 1, Algorithm 5)

The boxes of the list OldList are bisected until their width in all dimensions is lower than a specified precision parameter ε. This step is detailed in the Algorithm 4 (function Mince). For each dimension of the paving (lines 3,7,11, Algorithm 4), if the length is higher than the specified value ε, the box is bisected. A new box is then added at the end of the list of boxes and will be considered by the mince function later.

Calculating the inclusion function (line 2-4 Algorithm 5)

The images of the resulting boxes are then evaluated using the inclusion function for **f** and the Profil/Bias library. According to Eq. (2), the union of these images is guaranteed to contain $\mathbf{f}(X_{k-1|k-1},[\mathbf{u}+\mathbf{v}])$. When ε decreases, the image subpaving $X_{k|k-1}$ gets closer to the optimal subpaving. However the price to be paid is an increased prediction computing time.

Merging the subpaving (line 5-6 Algorithm 5)

The images are merged to get a subpaving guaranteed to contain the configuration of the vehicle (line 6, Algorithm 5). This last step builds a subpaving (a set of non-overlapping boxes) represented as a binary tree. It reduces the number of boxes that will have to be used in further steps. Furthermore, the list List is no longer useful and is deleted.

4.2. Estimation step

The estimation step integrates the data provided by the sonars of the experimental platform.

Algorithm 4 Mince all the boxes of a list

```
MINCE(list List, float $\varepsilon$)
1-  for each box $[[x]\,[y]\,[\theta]] \in$ List do
2-    while $w([x]) > \varepsilon$ or $w([y]) > \varepsilon$ or $w([\theta]) > \varepsilon$ do
3-      if $w([x]) > \varepsilon$ then
4-        addBoxToEndOfList($[\frac{\underline{x}+\overline{x}}{2}, \overline{x}] \times [\underline{y}, \overline{y}] \times [\underline{\theta}, \overline{\theta}]$,List)
5-        $[x] \leftarrow [\underline{x}, \frac{\underline{x}+\overline{x}}{2}]$
6-      end if
7-      if $w([y]) > \varepsilon$ then
8-        addBoxToEndOfList($[\underline{x}, \overline{x}] \times [\frac{\underline{y}+\overline{y}}{2}, \overline{y}] \times [\underline{\theta}, \overline{\theta}]$,List)
9-        $[y] \leftarrow [\underline{y}, \frac{\underline{y}+\overline{y}}{2}]$
10-     end if
11-     if $w([\theta]) > \varepsilon$ then
12-       addBoxToEndOfList($[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}] \times [\frac{\underline{\theta}+\overline{\theta}}{2}, \overline{\theta}]$,List)
13-       $[\theta] \leftarrow [\underline{\theta}, \frac{\underline{\theta}+\overline{\theta}}{2}]$
14-     end if
15-   end while
16- end for
```

Algorithm 5 Calculate the image of a SubPaving

```
tree IMAGESP ($f_{[]}$,list List, int $nt_r$, int $nt_l$,float $\varepsilon$)
1-  MINCE(List,$\varepsilon$)
2-  for each box $[\mathbf{x}]_{list}$ $\in$List do
3-      $[\mathbf{x}]_{list} \leftarrow \mathbf{f}_{[]}([\mathbf{x}]_{list}, nt_r, nt_l)$
4-  end for
5-  $[\mathbf{x}]_{bounding}$$\leftarrow$BOUNDING (List)
6-  Tree$\leftarrow$ BUILDSP($[\mathbf{x}]_{bounding}$,List,$\varepsilon$)
7-  Delete (List)
8-  return Tree
```

4.2.1. Sonar data prediction

The measurement equation at time k for the i-th sonar can then be described as

$$[y_{k,i}] = \mathbf{h}(\mathbf{s}_i([\mathbf{x}])) + [w_{k,i}], \tag{13}$$

with the noise measurement $[w_{k,i}] = [-\alpha\mathbf{h}(\mathbf{s}_i([\mathbf{x}])), \alpha\mathbf{h}(\mathbf{s}_i([\mathbf{x}]))]$ and $\mathbf{h}(\mathbf{s}_i([\mathbf{x}]))$ the smallest distance between the sensor positioned at $\mathbf{s}_i([\mathbf{x}])$ and the nearest segment located inside the emission cone (γ is the half aperture cone). The computing of $[\mathbf{r}_{\min}] = \mathbf{h}_{[]}(\mathbf{s}_i([\mathbf{x}]))$ is given in Algorithm 6. For each wall of the environment (see line 1,2,3, Algorithm 6), the smallest distance between the wall and a box is computed whilst taking into account the robot uncertainty in which the sensor is guaranteed to be. The segments are defined by their extremities (**a** and **b**). The aperture cone extremities are described by the two vectors $\mathbf{u}_1$ and $\mathbf{u}_2$. More details of this function are reported in [20], [12], [11].

All n_S measurement equations (13) may then be gathered to form $[\mathbf{r}_{\min}]$. Algorithm 6 allows an easy calculation of $[\mathbf{r}_{\min}]$ but it is a rather crude description of the range measurement process. Measurements resulting from multiple reflections of the transmitted wave, for instance, may not be explained by this model. Nevertheless, experimental results show us that such a simulation provide good enough localization results.

Each of the n_S sonars of the vehicle provides a range measurement. To the i-th measurement $y_{k,i}$, an interval $[y_{k,i}] = [y_{k,i} - \overline{w}_i, y_{k,i} - \underline{w}_i]$ may be associated. All $[y_{k,i}]$, measurements are gathered to form $[\mathbf{y}_k]$. The output $[\mathbf{y}_k]$, available at time k, can be taken into account by updating $X_{k|k-1}$ (the previous localization set obtained during the prediction step) into an outer approximation $X_{k|k}$

$$X_{k|k} = \mathbf{h}_k^{-1}([\mathbf{y}_k]) \cap X_{k|k-1}. \tag{14}$$

Since $X_{k|k-1}$ is described by a union of boxes, $X_{k|k}$ may be obtained as a union of boxes using the SIVIA (Set Inversion Via Interval Analysis) algorithm [11].

Algorithm 6 Sonar simulation

```
interval Sonar_simu (Wall Wall[nb_walls], box [s])
1- i←0
2- $[r_{min}] \leftarrow [+\infty, +\infty]$
3- $\overrightarrow{[\mathbf{u_1}]} \leftarrow calcul_{u_1}([s], \gamma)$
4- $\overrightarrow{[\mathbf{u_2}]} \leftarrow calcul_{u_2}([s], \gamma)$
5- for $w \leftarrow 0$ to nb_walls do
6-   a ←Wall[$w$].a
7-   b ←Wall[$w$].b
8-   $[t_1] \leftarrow \left(\det\left(\overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{a}[s]}\right) \geq 0\right)$
9-   if $(\underline{t_1} \geq 0)$ then return $(+\infty)$ end if
10-  $[t_h] \leftarrow \left(\left\langle\overrightarrow{\mathbf{ab}}, \overrightarrow{[s]\,\mathbf{a}}\right\rangle \leq 0\right) . \left(\left\langle\overrightarrow{\mathbf{ab}}, \overrightarrow{[s]\,\mathbf{b}}\right\rangle \geq 0\right)$
11-      $. \left(\left\langle\overrightarrow{\mathbf{ab}}, \overrightarrow{[\mathbf{u_1}]}\right\rangle \leq 0\right) . \left(\left\langle\overrightarrow{\mathbf{ab}}, \overrightarrow{[\mathbf{u_2}]}\right\rangle \geq 0\right)\right)$
12-  if $(\overline{t_h} = 1)$ then $[r_\mathbf{h}] \leftarrow \left\|\overrightarrow{[s]\,\mathbf{h}}\right\|$
13-      else $r_\mathbf{h} \leftarrow +\infty$
14-  end if
15-  $[t_a] \leftarrow \left(\left(\det\left(\overrightarrow{[\mathbf{u}_1]}, \overrightarrow{[s]\,\mathbf{a}}\right) \geq 0\right) . \left(\det\left(\overrightarrow{[\mathbf{u}_2]}, \overrightarrow{[s]\,\mathbf{a}}\right) \leq 0\right)\right)$
16-  if $(\overline{t_a} = 1)$ then $[r_\mathbf{a}] \leftarrow \left\|\overrightarrow{[s]\,\mathbf{a}}\right\|$
17-      else $r_\mathbf{a} \leftarrow +\infty$
18-  end if
19-  $[t_b] \leftarrow \left(\left(\det\left(\overrightarrow{[\mathbf{u}_1]}, \overrightarrow{[s]\,\mathbf{b}}\right) \geq 0\right) . \left(\det\left(\overrightarrow{[\mathbf{u}_2]}, \overrightarrow{[s]\,\mathbf{b}}\right) \leq 0\right)\right)$
20-  if $(\overline{t_b} = 1)$ then $[r_b] \leftarrow \left\|\overrightarrow{[s]\,\mathbf{b}}\right\|$
21-      else $r_\mathbf{b} \leftarrow +\infty$;
22-  end if
23-  $[t_{c1}] \leftarrow \left(\left(\det\left(\overrightarrow{[s]\,\mathbf{a}}, \overrightarrow{[\mathbf{u}_1]}\right) \geq 0\right) . \left(\det\left(\overrightarrow{[s]\,\mathbf{b}}, \overrightarrow{[\mathbf{u}_1]}\right) \leq 0\right)\right)$
24-  if $(\overline{t_{c_1}} = 1)$ then $[r_{\mathbf{c}_1}] \leftarrow \left\|\overrightarrow{[s]\,\mathbf{c}_1}\right\|$
25-      else $r_{\mathbf{c}_1} \leftarrow +\infty$
26-  end if
27-  $[t_{c_2}] \leftarrow \left(\left(\det\left(\overrightarrow{[s]\,\mathbf{a}}, \overrightarrow{[\mathbf{u}_2]}\right) \geq 0\right) . \left(\det\left(\overrightarrow{[s]\,\mathbf{b}}, \overrightarrow{[\mathbf{u}_2]}\right) \leq 0\right)\right)$
28-  if $(\overline{t_{c_2}} = 1)$ then $[r_{c_2}] \leftarrow \left\|\overrightarrow{[s]\,\mathbf{c}_2}\right\|$
29-      else $r_{\mathbf{c}_2} \leftarrow +\infty$;
30-  end if
31-  $[r_{min}] \leftarrow (\min([r_{min}], [r_h], [r_\mathbf{a}], [r_\mathbf{b}], [r_{\mathbf{c}_1}], [r_{\mathbf{c}_2}]))$;
32- end for
33- return $[r_{min}]$
```

4.2.2. SIVIA

Starting from the root of the binary tree provided by the BUILDSP algorithm during the prediction step, each box of the tree is evaluated using $[\mathbf{r}_{\min}] = \mathbf{h}_{[]}(\mathbf{s}_i([\mathbf{x}]))$ as following:

- If $[\mathbf{r}_{\min}] \subset [\mathbf{y}_k]$ then any $\mathbf{x} \in [\mathbf{x}]$ is thus consistent with the measurements and noise bounds and $[\mathbf{x}]$ is proved to be in $X_{k|k}$ (line 8 Algorithm 7). $[\mathbf{x}]$ is kept in the solution list.
- If $[\mathbf{r}_{\min}] \cap [\mathbf{y}_k] = \varnothing$ then there is no $\mathbf{x}$ in $[\mathbf{x}]$ that is consistent with the measurements and noise bounds and $[\mathbf{x}]$ does not belong to $X_{k|k}$ (line 11 Algorithm 7). $[\mathbf{x}]$ is eliminated.
- If $[\mathbf{r}_{\min}] \cap [\mathbf{y}_k] \neq \varnothing$ and if $[\mathbf{x}] < \varepsilon$, then the box is too little to be cut. At least one configuration in $[\mathbf{x}]$ is consistent with the measurements and noise bounds and due to its small dimension, $[\mathbf{x}]$ is kept (line 14 Algorithm 7).
- If $[\mathbf{r}_{\min}] \cap [\mathbf{y}_k] \neq \varnothing$ and if $[\mathbf{x}] \geq \varepsilon$, the same tests are applied to the right and left children (if they do not exist they are calculated).

4.2.3. Robustness to outliers

For ultrasonic range measurements, it sometimes occurs than more than 50% of the data collected turn out to correspond to outliers. These outliers may correspond to an outdated map of the environment, to sensor failures, to people moving in the environment, etc. As a consequence, the set characterized by SIVIA may very frequently turn out to be empty. To solve this problem, a robust variant (Algorithm 9) of the correction step has to be used. This variant keeps any $\mathbf{x} \in [\mathbf{x}]$ which is consistent with at least n_o measurements among the n_S available.

At the beginning of a correction step, the number of outliers is unknown and should be calculated. A first approach would be to characterize the set of all $\mathbf{x}_k$ that are

Algorithm 7 SIVIA

```
tree SIVIA(tree Tree, interval $[\mathbf{y}_k]$, float $\varepsilon$)
1-  if Tree= $\varnothing$ then return NULL end if
2-  $[\mathbf{x}] \leftarrow$Tree.root
3-  for $i \leftarrow 1$ to $n_s$ do
4-      $[\mathbf{s}] \leftarrow \mathbf{s}_i([\mathbf{x}])$
5-      $[r_{min}]_i \leftarrow$SONAR_SIMU (Walls[], $[\mathbf{s}]$)
6-  end for
7-  $[\mathbf{r}_{min}] \leftarrow [[r_{min}]_0, ..., [r_{min}]_{n_s}]$
8-  if $[\mathbf{r}_{min}] \subset [\mathbf{y}_k]$ then
9-      return Tree
10- end if
11- if $[\mathbf{r}_{min}] \cap [\mathbf{y}_k] = \varnothing$ then
12-     return $\varnothing$
13- end if
14- if $[\mathbf{x}] < \varepsilon$ then
15-     return Tree
16- end if
17- if Tree.child$_{left} = \varnothing$ and Tree.child$_{right} = \varnothing$ then
18-     Cut $[\mathbf{x}]$ into $[\mathbf{x}]_{left}$, $[\mathbf{x}]_{right}$ among the
                                   larger dimension
19-     Tree.child$_{left} \leftarrow [\mathbf{x}]_{left}$
20-     Tree.child$_{right} \leftarrow [\mathbf{x}]_{right}$
21- end if
22- Tree.child$_{left}$ = SIVIA(Tree.child$_{left}$, $[\mathbf{y}_k]$, $\varepsilon$)
23- Tree.child$_{right}$= SIVIA(Tree.child$_{right}$, $[\mathbf{y}_k]$, $\varepsilon$)
24- return Tree
```

Algorithm 8 Robust localization taken into account outliers

```
tree ROBUST_LOCALIZATION (tree Tree, interval $[\mathbf{y}_k]$, float $\varepsilon$)
1-  $\alpha \leftarrow 0$
2-  repeat
3-      NewTree$\leftarrow$Rob_SIVIA(Tree,$[\mathbf{y}_k]$,$\alpha$, $\varepsilon$)
4-      $\alpha \leftarrow \alpha + 1$
5-  until NewTree$\neq \emptyset$ and $\alpha <= n_s$
6-  $n \leftarrow$min($\alpha - 1 + security, n_s$)
7-  if $n = n_s$and NewTree= $\emptyset$ then
8-      return Tree
9-  end if
10- NewTree$\leftarrow$Rob_SIVIA(Tree,$[\mathbf{y}_k]$,$n$, $\varepsilon$)
11- return NewTree
```

consistent with α outliers, starting with α=0 and increasing α by one until the solution becomes non-empty (line 1-5 in Algorithm 8). Once such a nonempty set is obtained, and if the number α of outliers is smaller than n_S- *security*, one add a number of *security* outliers to increase robustness against undetected outliers (lines 6-10). However this causes deterioration in the precision of the localization. In order to deal with the outliers, the function SIVIA should have a new parameter n_o: the number of outliers that have to be considered. This alters the algorithm accordingly and leads to the Rob_SIVIA algorithm presented in Algorithm 9.

This new algorithm looks like the Algorithm 7, but it considers the sonar simulations one by one (line 5-14) to determine the number of consistent measurements (line 9) and erroneous measurements (line 11). If there are too many erroneous data, the tree is not a solution (line 15-17). If there are enough consistent data or if the minimum precision is reached (line 18-22), the tree is kept. If none of these cases occur then the current box is cut and the tests are repeated on each of the new boxes.

Finally, Algorithm 10 resumes the localization process using the presented algorithms.

Algorithm 9 Robust SIVIA

```
tree Rob_SIVIA(tree Tree, interval $[\mathbf{y}_k]$, int $n_o$, float $\varepsilon$)
1-  if Tree← $\varnothing$ then return NULL end if
2-  $[\mathbf{x}]$ ←Tree.root
3-  Sonar_ok← 0
4-  Sonar_bad← 0
5-  for $i \leftarrow 0$ to $n_s$ do
6-      $[\mathbf{s}] \leftarrow \mathbf{s}_i([\mathbf{x}])$
7-      $[r_{min}]$ ←Sonar_simu (Walls[], $[\mathbf{s}]$)
8-      if $([r_{min}] \subset [y_k])$ or
              $([r_{min}] \cap [y_k] \neq \varnothing$ and $[\mathbf{x}] < \varepsilon)$ then
9-          Sonar_ok←Sonar_ok+1
10-     end if
11-     if $[r_{min}] \cap [y_k] = \varnothing$ then
12-         Sonar_bad←Sonar_bad+1
13-     end if
14- end for
15- if Sonar_bad$\geq n_o$ then
16-     return $\varnothing$
17- end if
18- if Sonar_ok$\geq n_s - n_o$ then
19-     return Tree
20- end if
21- if $[\mathbf{x}] < \varepsilon$ then
22-     return Tree
23- else if Tree.child$_{left} = \varnothing$ and Tree.child$_{right} = \varnothing$
24-     then Cut $[\mathbf{x}]$ into $[\mathbf{x}]_{left}$, $[\mathbf{x}]_{right}$ among the
                                              larger dimension
25-     Tree.child$_{left} \leftarrow [\mathbf{x}]_{left}$
26-     Tree.child$_{right} \leftarrow [\mathbf{x}]_{right}$
27- end if
28- Tree.child$_{left}$=Rob_SIVIA(Tree.child$_{left}$, $[\mathbf{y}_k]$, $n_o$, $\varepsilon$)
29- Tree.child$_{right}$=Rob_SIVIA(Tree.child$_{right}$, $[\mathbf{y}_k]$, $n_o$, $\varepsilon$)
30- return Tree
```

Algorithm 10 Localization process

```
1- while true do
2- Tree.root←initial_box_of_configuration
// prediction step
3-  repeat
4-      List ←TREE_TO_LIST (Tree)
5-      Delete (Tree)
6-      Tree ← IMAGESP ($f_{[]}$,List, $nt_r$, $nt_l$, $\varepsilon$)
7-  until available(sonar_measurements)
// estimation step
8-  $\mathbf{y}_k$ ←sonar_measurements
9-  $[\mathbf{y}_k] \leftarrow \mathbf{y}_k + [\mathbf{w}_k]$
10- Tree ← ROBUST_LOCALIZATION (Tree,$[\mathbf{y}_k]$, $\varepsilon$)
11- end while
```

5. Complexity study

In this section, the complexity of the localization process is studied according to each algorithm presented in the precedent parts.

5.1. Prediction step

During the prediction (Algorithm 10), the functions TREE_TO_LIST and IMAGESP are used.

The TREE_TO_LIST function complexity

This function iterates over the $n_{boxes_{tree}}$ nodes of the tree and creates a list of $n_{boxes_{leaves}}$ elements (with $n_{boxes_{tree}} = n_{boxes_{leaves}} \times 2-1$ in the worst case scenario). Thus the complexity of this algorithm is $O(n_{boxes_{tree}})$.

The IMAGESP function complexity

IMAGESP is summarized in Algorithm 5 and uses the functions MINCE and BUILDSP.

The MINCE function complexity

The MINCE function (Algorithm 4) starts from a list of n_{list} elements and bisects these boxes into 2 boxes until their dimensions become smaller than e. In the worst case, the number of operations is equal to the number of nodes $n_{boxes_{tree}}$ of a binary tree with $n_{boxes_{leaves}}$ leaves. Thus, the complexity of this algorithm is $O(n_{boxes_{tree}})$.

The BUILDSP function complexity

BUILDSP, presented in Algorithm 2, can be summarized as:

- line 2 calls the function Bounding (Algorithm 3) which iterates over the list of boxes ($O(n_{boxes_{leaves}})$)
- lines 10-12 and lines 17-24 show two iterations over the list of boxes ($O(2\times n_{boxes_{leaves}})$)
- lines 25-26, two recursive calls of BUILDSP
- lines 27-28, the lists are deleted, two iterations over the list of boxes ($O(2\times n_{boxes_{leaves}})$)

Consequently, the list of boxes is visited 5 times at each recursive call. The recursive calls are done until the $n_{boxes_{leaves}}$ leaves of the tree are reached. Consequently, the number of recursive calls is:

$$n_{calls_{lBuildSp}} = 1+2+4+...+n_{boxes_{leaves}} = n_{boxes_{tree}} \qquad (15)$$

Two lists of boxes are created by BUILDSP. Each list contains the boxes that cover a part of the considered space. In the worst case, each list contains all the boxes of the initial list.

The complexity of the BUILDSP function is $O(n_{boxes_{tree}} \times 5\times n_{boxes_{leaves}})$ which can be simplified to $O(n^2_{boxes_{tree}})$.

Complexity of the prediction step

According to the Algorithm 5:

- line 1 uses the function Mince (Algorithm 4) that has a complexity of $O(n_{boxes_{tree}})$,
- lines 2-4, iterate over the list of boxes (complexity of $O(n_{boxes_{tree}})$),
- line 5 uses the function Bounding (Algorithm 3), which iterates over the list of boxes (complexity in $O(n_{boxes_{leaves}})$),
- line 6 uses the function BUILDSP, which has a complexity of $O(n^2_{boxes_{tree}})$,
- Line 7: the list is deleted ($O(n_{boxes_{leaves}})$).

In addition, according to Algorithm 10, during prediction step a call of TREE_TO_LIST and delete(tree) is done. Each of them has a complexity in $O(n_{boxes_{tree}})$.

Knowing that $n_{boxes_{tree}} = 2n_{boxes_{leaves}} - 1$, we can replace $n_{boxes_{leaves}}$ by $n_{boxes_{tree}}$ in the above complexity. Then, we can notice that the main part of the prediction step complexity is due to BUILDSP. Finally, the overall complexity of the prediction step is $O(n^2_{boxes_{tree}})$.

5.2. Estimation step

During the estimation step (see Algorithm 10), the function Robust_localization (Algorithm 8) is called upon. Next, Robust_localization uses the function Rob_SIVIA. Rob_SIVIA may be broken down into 2 main steps:

- line 5-14, n_s calls of sonar simulation,
- lines 25-26, two recursive calls of Rob_SIVIA.

The number of recursive calls of the function Rob_SIVIA depends on the number of boxes of the binary tree. Thus Rob_SIVIA will be called a maximum of $n_{boxes_{tree}}$ times and the complexity of Rob_SIVIA is in $O(n_s \times n_{boxes_{tree}} \times n_{walls})$ (see Algorithm 6), where n_{walls} is the number of the walls. As Rob_SIVIA is called n_s time by the Robust_localization function, the complexity of the estimation step is in $O(n_s^2 \times n_{walls} \times n_{boxes_{tree}})$.

6. Toward a real-time computing

6.1. First improvement

The BUILSP algorithm has high complexity and is realised during each prediction step *ie* ten to a thousand times between each estimation step. The goal of the BUILSP algorithm is to reduce further calculation (to decrease the number of boxes and to eliminate overlapping boxes). Nevertheless, the use of BUILSP is not mandatory in the prediction step.

To decrease the calculation time of the prediction step, the BUILSP algorithm was moved outside of the prediction step and was only done before each estimation step. This means that BUILSP is moved after the prediction loop *ie* from the IMAGESP algorithm to between line 7 and 8 of Algorithm 10. It reduces the prediction step complexity to $O(n_{boxes_{tree}})$ and increases the estimation step complexity to $O(n^2_{boxes_{tree}} + n_{boxes_{tree}} \times n_s^2 \times n_{walls})$. Jointly with this modification, the TREE_TO_LIST step and the deletion of the tree are moved after each estimation step (rather than before each prediction step). In return, the number of boxes to be taken into account during the prediction step increases, and overlapping boxes are present. However, the whole calculation time decreases during the prediction step as well as during the complete localization process. Such an improvement was already applied in [26, 25, 14].

6.2. Tuning of the parameter ε

The number of nodes $n_{boxes_{tree}}$ of the binary tree is the biggest when the number of leaf nodes $n_{boxes_{leaves}}$ is the biggest. In such a case the tree is well balanced. $n_{boxes_{leaves}}$ is the biggest when the leaf nodes have the smallest dimensions (lower than ε).

Assuming that the bounding box of the studied set is $[\mathbf{x}]_{inc} = [x]_{inc} \times [y]_{inc} \times [\theta]_{inc}$ and that ε defines the precision parameter (any dimension of any box can be divided by 2 until its size is lower than ε), the maximum number of boxes $n_{boxes_{leaves}}$ is:

$$n_{boxes_{leaves}} = \left\lceil \frac{w[x]_{inc}}{\varepsilon/2} \right\rceil \times \left\lceil \frac{w[y]_{inc}}{\varepsilon/2} \right\rceil \times \left\lceil \frac{w[\theta]_{inc}}{\varepsilon/2} \right\rceil, \qquad (16)$$

where $\left\lceil \frac{w[x]_{inc}}{\varepsilon/2} \right\rceil$ is the smallest integer greater or equal to $\frac{w[x]_{inc}}{\varepsilon/2}$ and $w[x]_{inc}$ is the width of the box $[x]_{inc}$.

We can experimentally determine the maximum number $n^{\max}_{boxes_{leaves}}$ of boxes allowed to achieve a real time computing. Next we need to maintain a number of boxes lower or equal to $n^{\max}_{boxes_{leaves}}$ in order to maintain a real time:

$$n_{boxes_{leaves}} \leq n^{\max}_{boxes_{leaves}} \qquad (17)$$

Thus, ε can be deduced from:

$$\left(\frac{w[x]_{inc}}{\varepsilon/2}+1\right) \times \left(\frac{w[y]_{inc}}{\varepsilon/2}+1\right) \times \left(\frac{w[\theta]_{inc}}{\varepsilon/2}+1\right) \leq n^{\max}_{boxes_{leaves}}, \qquad (18)$$

or equivalently by the following cubic function:

$$(1 - n^{\max}_{boxes_{leaves}})\, \varepsilon^3 + 2\varepsilon^2(w[x]_{inc}+w[y]_{inc}+w[\theta]_{inc}) +$$
$$+ 4\varepsilon(w[x]_{inc}w[y]_{inc}+w[y]_{inc}w[\theta]_{inc}+w[x]_{inc}w[\theta]_{inc}) +$$
$$+8w[x]_{inc}w[y]_{inc}w[\theta]_{inc} \leq 0, \qquad (19)$$

Equation (19) has at least one solution among the real numbers. We retain the smallest value of ε if there is more than one solution.

By considering that the three added boxes (the three "1") in Eq. (18) do not significantly modify the calculation time, we obtain the following approximation:

$$\varepsilon \cong \sqrt[3]{\frac{8w[x]_{inc} \times w[y]_{inc} \times w[\theta]_{inc}}{n^{\max}_{boxes_{leaves}}}}, \qquad (20)$$

We can now adapt the ε value to the embedded computing power. Unfortunately, a high ε value leads to a poor (imprecise) localization. This is the price paid for having a slow embedded computer.

7. Experimental validation

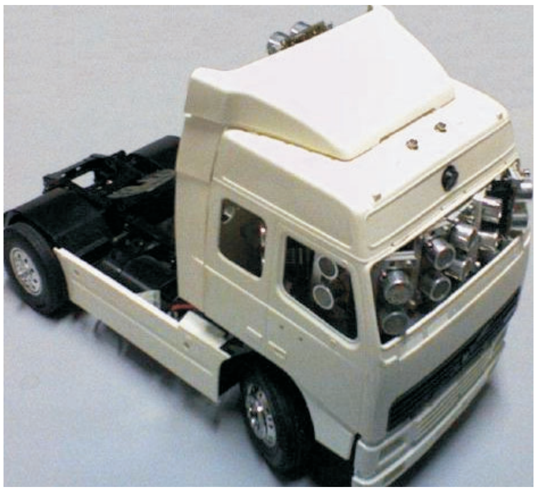
Minitruck (Figure 2-a) is a generic electric vehicle developed in the laboratory and equipped with an embedded electronic system, actuators and sensors. This multisensor platform has sufficient computing power to execute the algorithm autonomously. Nevertheless, a Client/Server architecture with wireless communication enables high computing power calculations to be sent to a distant computer. It allows the user to quickly add his own algorithms so as to test his theoretical works.

7.1. Introduction

An experimental platform is necessary for an experimental validation of algorithms. However, real scale platform is generally expensive to buy and maintain and moreover time-consuming [32, 4]. Most of the platforms based on a reduced model, do not support time consuming algorithms [31].

The platform *Minitruck* (Figure 2-a) does not have these disadvantages, the 400 Mhz Intel Xscale processor (480 Mips) allowing the user to run consuming computing programs directly on the platform. Furthermore, the possibility of accessing a distant computer still increases the computing power of the platform. In addition, Minitruck has advantages of low development cost, low maintenance demand, a small size and reduced weight. There is also the possibility of doing indoor as well as outdoor experiments with a low electrical consumption.

(a) Minitruck



(b) The sensors

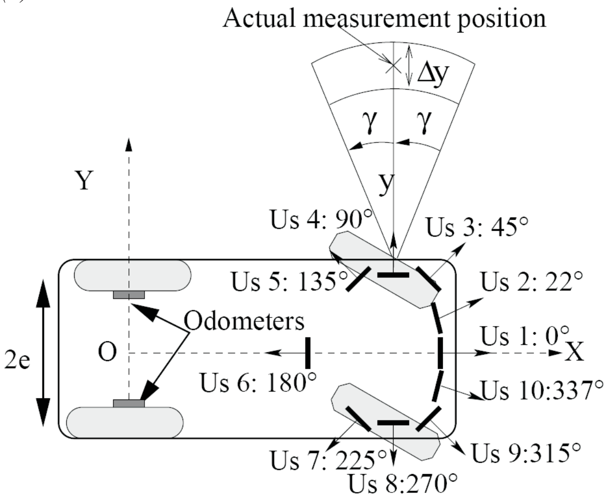


Fig. 2. Location of the sensors and uncertainty of the sonar measurements.

The platform embeds different kinds of sensors: ultrasound sonars, odometers and a camera with the possibility of easily adding other sensors. A client-server architecture was chosen with a powerful distant workstation. The communication is ensured by a wireless connection and a high level protocol that we defined. All the vehicle actuators and sensors are accessible through the wireless network. The vehicle also embeds a data server which can accept requests and commands from one or more distant workstations.

7.2. Experimental platform

7.2.1. Mechanical considerations

To reduce the development cost of the platform, an existing mechanical base of a vehicle model reduced to 1/20 was used. With small size, 50 cm length by 22 cm width, and a weight of less than 4 kg, the platform allows easy experimentation indoors as well as outdoors. The experimental platform can easily be carried everywhere. The propulsion is provided by an electric motor (D.C. motor) assembled on a reducer and a differential bridge on the aft wheels. The vehicle is able to go forward and backward and can reach speeds of up to 10 meters per second because of a three gears gearbox. The vehicle direction control is done by a numerical servo-motor on the front wheels. It is able to turn the wheels from -35° to +35°. Two other servo-motors are used to control the motor speed through a power module and the gearbox.

7.2.2. Electronic considerations

For the power supply of the vehicle, two batteries NiMh of 3300 mAh are used : one is soley for the electric motor and the other is used for electronic purposes. During our experiment, the autonomy of the vehicle was about an hour depending on the speed and the command to the servo-motors.

The embedded computer is based on an Intel Xscale PXA270 processor which has a 32 bits core clocked at 400 mHz. Sold by the GUMSTIX company (www.gumstx.com), it provides a complete Linux system based on the 2.6 Linux kernel. It integrates 128 MB SD-RAM, 4 MB strataflash and the possibility to add a MMC card (Multi Media Card). It has several I/O peripherals (80 I/O pins including 2 serial links, a wi-fi connection, I2C bus, SPI bus, USB, etc.). In addition, another processor, an Atmel 128, is available and is used to connect the incremental coders and the servo-motors. Figure 3 gives a synopsis presenting how the various embedded modules are connected to the Intel Xscale and the Atmel. The embedded computing power is sufficient to implement on-board basic tasks like collision avoidance, Kalman filtering localization, path tracking, automatic parking [27], etc. Tasks needing more computing power can be deported on a distant computer using a client-server architecture. In this case, the exchanged information is reduced to the sensor data from the experimental platform and to the command from the distant computer.

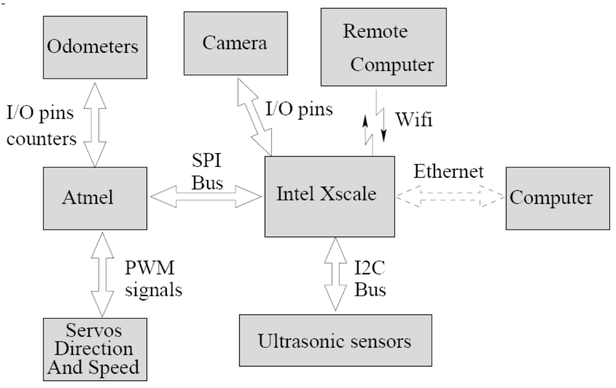


Fig. 3. Flowchart of the hardware.

7.2.3. Proprioceptive sensors

The digital optical coders HEDS 5540, developed by Hewlett-Packard, have a 500 points per turn angular resolution. Basically, a beam goes through a disc (Figure 4), and two sensors detect the beam crossing. These two sensors are needed to find the rotation direction. Counting the electric crenels, nt_r for the right wheel and nt_l for the left wheel deduces the angular displacement, see Eq. (8). Due to the vehicle wheel radius of r=4 cm, incremental coders give a half-millimeter-length precision, which is considered sufficient given the mechanical precision. They give information about the vehicle movements. These movements are measured by an incremental coder assembled on each rear wheel unit.

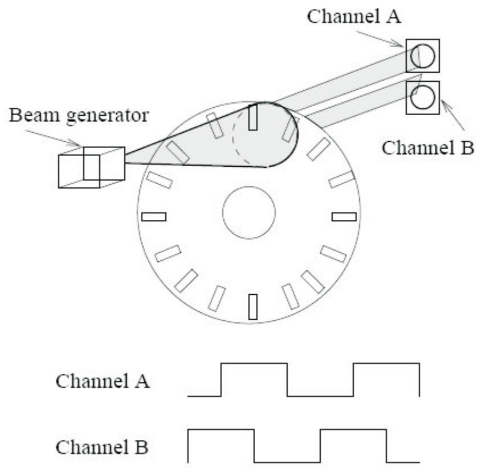


Fig. 4. An incremental coder.

7.2.4. Exteroceptive sensors

Overview

The platform is equipped with ten ultrasound sonars that give back information about the distance between them and any obstacle located in their aperture cone. The Polaroid Ultrasonic Ranging module is popular in the field of robotic sensor technology. Both its high current consumption (150mA) and its large size are not well adapted for a reduced model platform. The SFR08 of Devantech (see Figure 5) were chosen because of their small size (43x20x17 mm), their price (less than 60 Euros), their low consumption (50 mA during the shoot and 3 mA in standby) and because they integrate all the electronic treatments. Fixed around the vehicle, nine of the sensors are in the truck cab and sweep the front and the sides of the vehicle. A tenth sensor is placed on the rear of the cab and sweeps the rear of the vehicle. The sensor's orientation and position are presented on Figure 2b).
Minimum sensing distance

Each ultrasonic module uses a transmitter resonator and a receiver resonator, both calibrated at a frequency of 40 kHz. Considering the Fresnell zone [18], a close field in which high wave perturbation makes it highly improbable to detect any obstacle, we find:

$$l_0 = \frac{D^2}{4\lambda} = \frac{D^2.f}{4v} = \frac{(16\ mm)^2.40\ kHz}{4\times 340\ m.s^{-1}} = 7.5\ mm, \tag{21}$$

where D=16 mm is the sensors diameter, $\lambda = \frac{v}{f} = 8.65\int mm$

is the wavelength, v=340 m.s^{-1} the sound speed and f=40 kHz is the resonator frequency. Thus, the sensor will not be able to detect any obstacles below l_0=7.5 mm.

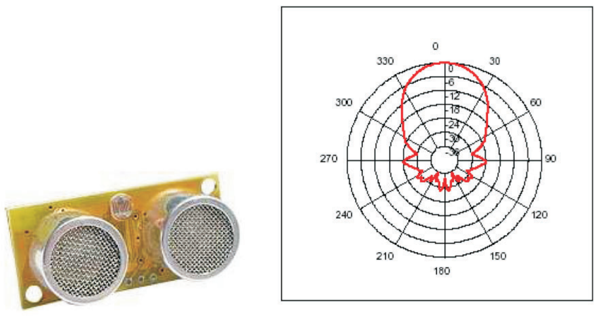


Fig. 5. SFR08 sensor and angle beam.

During the emission, the SFR08 send out an eight cycle burst at 40 kHz and is not able to detect any echo because the sensor processor does not check the reception resonator while it commands the emission resonator. Due to the sound speed, we can deduce that the sensor is blind below $\left(\frac{8}{40\ kHz}\times 340\ m.s^{-1}\right)/2 = 3.4\ cm$ (including the Fresnell zone).

In practice, the smallest distance the SFR08 succeeded in detecting is 2.9 cm between the sensor and an obstable. To this distance, we have to add the distance between the external grid of the resonator and the membrane: 0.5 cm. Thus, the minimum distance measured by the sensor is 3.4 cm. If the obstacle is closer than 3.4 cm then the calculated distance is at least 3.4 cm. In fact the SFR08 consider the first echo received after a time that corresponds to 3.4 cm. Practice and theory give the same result. Consequently, the measurement returned by the sonar simulator (see Section 5.2) should be bigger than or equal to 3.4 cm.

Maximum sensing distance

During the reception, the SFR08 can listen to an echo for a maximum of 65 ms. Its maximum sensing distance is (65 ms*346 m/s)/2=11 m. Experimentation was done to find the maximum distance the sensor is able to detect. To this purpose, we proceeded to take samples of 100 measurements, increasing the distance gradually. At 5 meters, the sonar could never detect the obstacle. The waiting time set by the manufacturer provides a maximum sensing distance of 11 meters, much further than the 5 meters the SFR08 is actually capable of. We chose to reduce this waiting time to 35 ms, which gives a theoretical maximum sensing distance of 5 meters, to reduce the measurement time. Consequently, the measurement returned by the sonar simulator should be lower than or equal to 5 m.

Measurement errors

Several experiments were done to find the sensor's properties. The sensor was placed at 200 cm from the ceiling of the room to approach ideal conditions and we proceeded to 1000 measurements.

The distribution gives an average of 200 cm with an error of 2 cm (1%), the repartition of these data is shown in Figure 6.

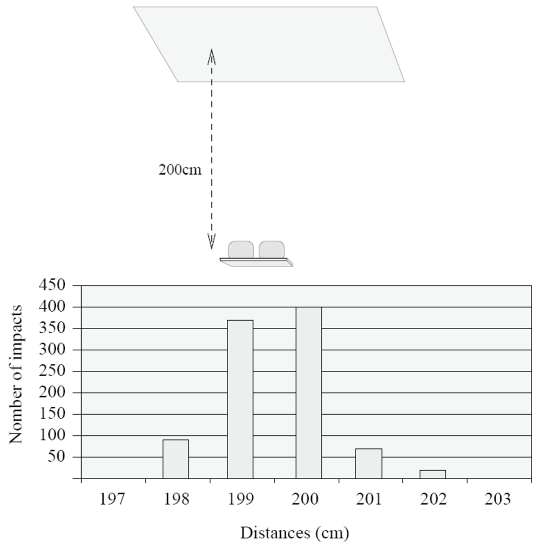


Fig. 6. Sonar data distribution.

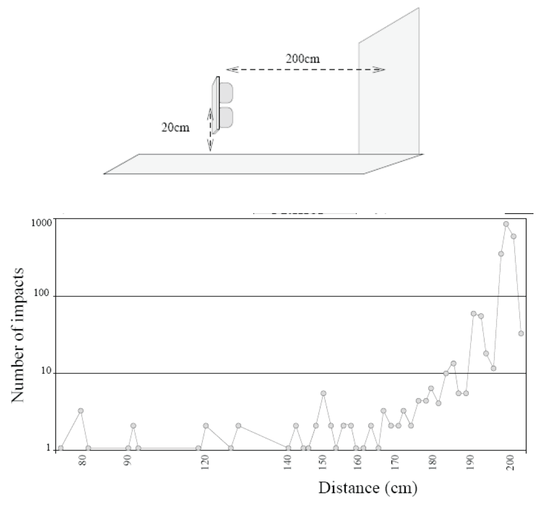


Fig. 7. Other experiment.

In ideal conditions, the sensors detect the distance with an error of about ± 1% (Figure 6). In addition to this error, the temperature variation in the sound speed causes an error in calculating the distance. Eq. (22) gives the sound speed v_{sound} depending on the sound speed at 0°C (v_0=331.3 m.s^{-1}) and on the temperature of the environment t in degrees Celsius.

$$v_{sound} = v_0 . \sqrt{1+\frac{t}{273}}, \qquad (22)$$

If we consider a temperature of 20°C in the calculation and if the real temperature is 25°C, an error of 1% on the result has to be added. Thus, the total uncertainty Δy (Figure 2) on the distance measurements y may be considered to be 2% of the result.

In the second experiment we considered real conditions: the vehicle was placed on the ground at a distance of 200 cm from a wall. Figure 7 presents a distribution in logarithmic scale of 2000 sonar measurements. We notice that several shorter distances were detected by the sensor. They are mainly due to reflections from the ground. Overall, almost 100 measurements were done under 190 cm. That means that 5% of measurements have an error of more than 5%. These experiments were done several times, showing that the data calculated by the sensor largely depends on the environment (floor, walls). The measurement error is set to 6% (5% + 1% in order to take into account the temperature change). Thus, α=6% in Equation (13).

Aperture cone

An obstacle is detected if it is in the emission cone of the sensor. This aperture cone depends on the incidence angle between the sensors and the object, the material of the object and its distance from the sensor. Experimentation showed that a maximal half aperture cone γ of 22° can be considered. It is the largest angular deviation for which a useful signal could be collected. This is consistent with the 3 dB beam width that can be obtained from Figure 5.

7.3. Experimental results

Using data obtained from the experimental platform Minitruck, the bounded error state estimation method was employed to solve the localization problem. The model uncertainties (Equations (9), (10) and (13)) are taken as Δr=5 mm, Δe=5 mm and α=0.05 . The ε parameter was automatically calculated using Eq. (20).

During this experiment, Minitruck moved in a corridor of our laboratory at a low speed (10 cm/s) (see Figure 8).

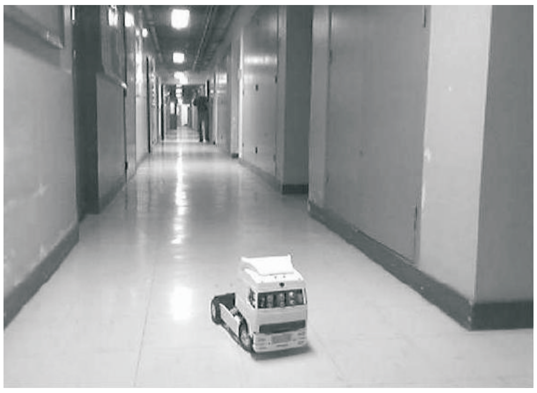


Fig. 8. Minitruck in its environment.

The curved solid line in Figure 9 represents the vehicle trajectory as provided by the odometric data alone. This predicted trajectory deviates from the actual trajectory. Such a localization error is due to an error in the initial configuration, the wheels slipping on the ground during

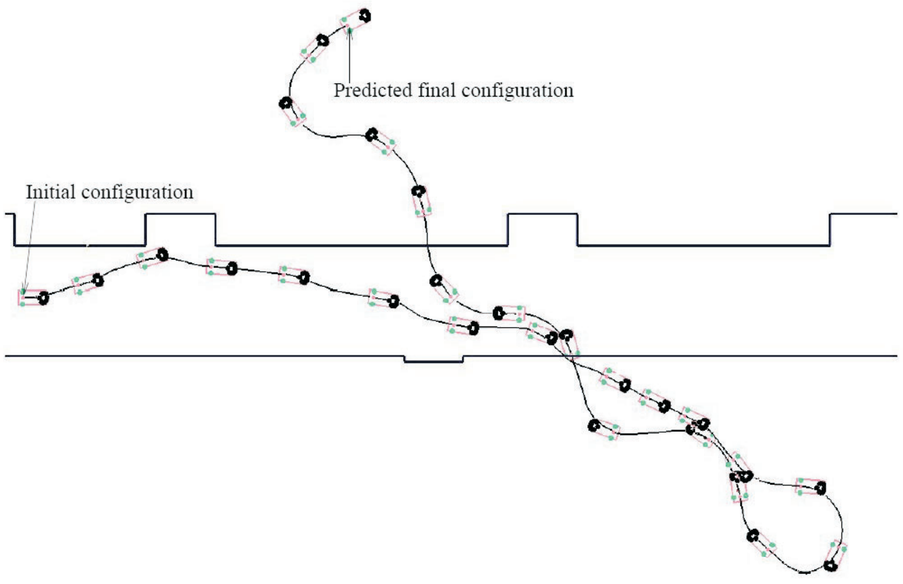
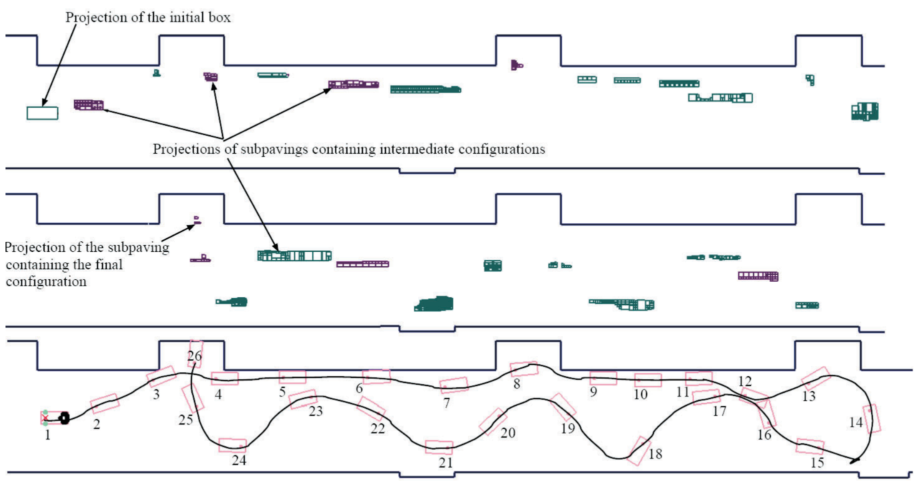


Fig. 9. Position of Minitruck as evaluated from odometric data alone.


Projection of the initial box
Projections of subpavings containing intermediate configurations
Projection of the subpaving containing the final configuration
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
Fig. 10. Projections onto the (X,Y)-plane of 26 subpavings.

the experiment and inaccurate values of the distance between the rear wheels and of their diameter. On a forty-meter course, the error on the arrival point is almost four meters.

Sonar measurements were collected each second. Unfortunately, a large part of this data has to be considered as erroneous and does not correspond to any charted element. The presence of outliers is due not only to the uncertainty of the results returned by the sensors but also to foreign elements not represented on the map which were in the field of the sensors at the time of measurements, such as people walking in the corridor and uncharted elements.

Thus, the correction step should only consider the relevant part of telemetric measurements, and be robust to the presence of outliers. In average, about 20 to 30 % of the measurements had to be considered as erroneous.

In this experiment, the initial configuration of *Minitruck* is only known to satisfy $x \in$ [26,0 m;26,5 m], $y \in$ [0,8 m;1,0 m] and $\theta \in$ [350,10].

Figure 10 displays the projection onto the (X,Y)-plane of 26 estimated configuration subpavings taken during the tracking of *Minitruck*. Despite the large amount of erroneous data, a relatively precise localization is provided, often with a precision of about 20 cm.

At the end of this experiment, the final configuration is well estimated (Table 1). The actual configuration is represented on Figure 10 and contained in the last subpaving provided.

Table 1. Actual configuration and estimated configuration subpaving provided by the bounded-error localization technique.

	Actual configuration	Estimated configuration
X (cm)	28,76	[28,76 ; 28,86]
Y (cm)	1,81	[1,71 ; 1,81]
θ (deg)	84	[74;85]

These results show the ability of the bounded-error localization technique to work in real time in a real environment.

8. Conclusion

In this paper, we presented in detail the bounded-error implementation and algorithms used to localize a mobile robot equipped with sonars. The algorithms of IMAGESP and BUILDSP for the prediction step, and the algorithms of SIVIA for the estimation step were provided. A complexity study of these algorithms was carried out. Furthermore, we gave a tuning of the parameter that allows us to achieve a real time localization. The final experiment provides results that show the ability of the bounded-error state estimation to solve the localization problem in real time. Further works will concern an acceleration of the estimation step (the most time consuming part of the algorithm) so as to enhance the localization precision on the slowest of computers.

List of symbols

$f_{[\,]}([x])$	Inclusion function that denotes the set of all values taken by $f(.)$ over $[x]$
$\mathbf{x} = (x,y,\theta)^T$	Vehicle configuration in the global frame
$L[\mathbf{x}]$, $R[\mathbf{x}]$	Respectively the right and the left children of $[\mathbf{x}]$, where $[x]$ is a node of a tree
$[\mathbf{x}]_{root}$	Box that contains all the boxes of the List (or of the Tree)
$X_{k\|k-1}$	The set that is consistent with sensor data provided at time $\underline{k}$ knowing $k-1$
ε	Precision parameter used to bisect a box
$\mathbf{s}_i([\mathbf{x}])$	Position of the sensor i when vehicle is in a configuration $[\mathbf{x}]$
γ	Half aperture cone of the sonar
$\mathbf{y}_k$	Gathered measurements for all $\lvert y_{k,i} \rvert$ sonar measurements at time k.

AUTHORS

Emmanuel Seignez* - Ecole Supérieure d'Ingénieurs en Electronique et Electrotechnique, 14 Quai de la somme, 80000 Amiens, tel.: +33 3 22 66 20 54, fax number: +33 3 22 66 20 00. E-mail: seignez@esiee-amiens.fr.
Alain Lambert - Institut d'Electronique Fondamentale Univ Paris Sud, CNRS, F-91405 Orsay, France . E-mail: alain.lambert@u-psud.fr.
* Corresponding author

References

[1] http://www.ti3.tu-harburg.de/software/profilenglisch.html.

[2] M. Arulampalam, S. Maskell, N. Gordon, T. Clapp, "A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking", *IEEE Transactions Signal Processing*, no. 50(2), 2002, pp. 174-188.

[3] J. Borenstein, B. Everett, L. Feng. *Navigating Mobile Robots: Systems and Techniques*. A.K. Peters Ltd., Wellesley, MA, 1996.

[4] S. Bouaziz, M. Fan, A. Lambert, T. Maurin, R. Reynaud, "Picar: experimental platform for road tracking applications". In: *Proceedings of the IEEE International Conference on Intelligent Vehicle (IV)*, 2003, pp. 495-499.

[5] W. Burgard, D. Fox, D. Hennig, T. Schmidt, "Estimating the absolute position of a mobile robot using position probability grids". In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, vol. 2, Portland, OR, 1996, pp. 896-901.

[6] W. Burgard, D. Fox, S. Thrun, "Active mobile robot localization". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

[7] C.K. Chui, G. Chen, *Kalman filtering with real-time applications*, Springer Verlag, 1990.

[8] I. Cox *et al.*, *Autonomous Robot Vehicles*, Springer-Verlag, 1990.

[9] J.L. Crowley, "World modeling and position estimation for a mobile robot using ultrasonic ranging". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, Scottsdale, AZ, 1989, pp. 674-680.

[10] K. Ito, K. Xiong, "Gaussian filters for nonlinear filtering problems". In: *IEEE Transactions on Automatic Control*, vol. 45, May 2000, pp. 910-927.

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

[12] M. Kieffer, "*Estimation ensembliste par analyse par intervalles, application a la localisation d'un véhicule*". PhD thesis, Université Paris-Sud, Orsay, France, 1999. (in French)

[13] M. Kieffer, L. Jaulin, E. Walter, D. Meizel, "Robust autonomous robot localization using interval analysis", *Reliable Computing*, no. 6(3), 2000, pp. 337-362.

[14] M. Kieffer, E. Seignez, A. Lambert, E. Walter, T. Maurin, "Guaranteed robust nonlinear state estimator with application to global vehicle tracking". In: *Proceedings of the IEEE International Conference on Decision and Control (CDC)* , Spain, 2005, pp. 6424-6429.

[15] O. Knuppel, "Bias-basic interval arithmetic subroutines. Technical report", *Technical Report 93.3*, Harburg-Hamburg, Germany, 1993.

[16] O. Knuppel, "Profil-programmers runtime optimized fast interval library. Technical report", *Technical Report 93.4*, Harburg-Hamburg, Germany, 1993.

[17] A. Lambert, N. Le Fort-Piat, "Safe task planning integrating uncertainties and local maps federation", *International Journal of Robotics Research*, no. 19(6), 2000, pp. 597-611.

[18] A. Lambert, Y. Pralus, J. Rivenez, "Ultrasons, propagation des ondes ultrasonores". In: *Centre Technique des Industries Mécanique*, 1990. (in French)

[19] J.J. Leonard, H.F. Durant-Whyte, "Mobile robot localization by tracking geometric beacons", In: *IEEE Transactions on Robotics and Automation*, no. 7(3), 1991, pp. 376-382.

[20] O. Lévęque. "*Méthodes ensemblistes pour la localisation de véhicules*". PhD dissertation, Université de Technologie, Compičgne, France, 1998. (in French)

[21]] Léveque, L. Jaulin, D. Meizel, E. Walter, "Vehicle localization from inaccurate telemetric data: a set inversion approach". In: *Proceedings of 5th IFAC Symposium on Robot Control, SY.RO.CO.'97*, vol. 1, Nantes, France, 1997, pp. 179-186.

[22] D. Meizel, O. Leveque, L. Jaulin, and E. Walter. Initial localization by set inversion. *IEEE Transactions on Robotics and Automation*, 18(6):966971, December 2002.

[23] R.E. Moore, *Interval Analysis*, Prentice-Hall: Englewood Cliffs, NJ, 1966.

[24] R.E. Moore, *Methods and Applications of Interval Analysis*, SIAM: Philadelphia, PA, 1979.

[25] E. Seignez, M. Kieffer, A. Lambert, E. Walter, T. Maurin, "Experimental vehicle localization by bounded-error state estimation using interval analysis". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, Canada, 2005, pp. 1277-1282.

[26] E. Seignez, M. Kieffer, A. Lambert, E. Walter, T. Maurin, "Real-time bounded-error state estimation for vehicle tracking", *International Journal of Robotics Research*, no. 28(1), 2009, pp. 34-48.

[27] E. Seignez, A. Lambert, T. Maurin, "Autonomous parking carrier for intelligent vehicle". In: *Proceedings of the IEEE International Conference on Intelligent Vehicle (IV)*, Las Vegas, USA, 2005, pp. 411-416.

[28] E. Seignez, A. Lambert, T. Maurin, « An experimental platform for testing localization algorithms". In: *Proc. of the IEEE International Conference On Information And Communication Technologies: From Theory To Application, ISBN: 0-7803-68483-0*, Siria, 2006.

[29] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, D. Schulz, "Minerva: A second generation mobile tour-guide robot". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 5, 1999, pp. 1999-2005.

[30] S. Thrun, D. Fox, W. Burgard, F. Dellaert, "Robust monte carlo localization for mobile robots", *Artificial Intelligence Journal*, no. 128(1-2), 2001, pp. 99-141.

[31] R.W. Wall, J. Bennett, K. Lichy, E. Owings, G. Eis, "Creating a low cost autonomous vehicle". In: *IEEE Internationnal Conference of the Industrial Electronics Society (IECON)*, vol. 4, Sevilla, Spain, 2002, pp. 3112-3116.

[32] E. Woo, B.A. MacDonald, F. Trépanier, "Distributed mobile robot application infrastructure". In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, Las Vegas, Oct. 2003, pp. 1475-1480.