Mobile Robot Start-Up Positioning using Interval Analysis[*]

Antonio B. Martínez, Josep Escoda and Antonio Benedico

Departament d'Enginyeria de Sistemes, Universitat Politècnica de Catalunya (ESAII-UPC)
Campus Sud UPC – Edifici U
C/ Pau Gargallo, 5 – 08028 Barcelona, Spain

E-mail: {Antonio.B.Martinez,Josep.Escoda,Toni.Benedico}@upc.edu

***Abstract* – This paper shows how to adapt the Set Inversion Via Interval Analysis algorithm (SIVIA) to the problem of mobile robot start-up positioning with a goniometric sensor. This approach is an alternative way to Markov, MonteCarlo and Neural Networks techniques, among others, that deal with that problem. SIVIA brings a global solution and confines the desired result into a union of intervals.**

Index Terms – start-up positioning, mobile robot, interval analysis.

I. Introduction

The goal of this paper is to present an approach to the problem of start-up positioning of a mobile robot using interval analysis techniques. In order to use this methodology, working environment must contain a sufficient number of landmarks, and a map of the total workspace must be provided. With bearing information obtained from goniometric sensor and knowledge of landmarks position, it can be obtained the absolute pose of the robot. By absolute pose, it must be understood the [x,y,Ψ] triplet: [x,y] position plus the orientation Ψ with respect to workspace framework.

Triangulation is the method used traditionally to position a mobile robot with bearing information with respect to three known landmarks [1]. However, these three landmarks must have been identified *a priori*. In the case that no identification information is known, as in the case of start-up positioning, triangulation can not be used directly.

Moreover, eventually some added problems may arise:

1) Occlusions: A landmark may eventually be occluded by a column or other vehicle.

2) Outliers: Depending on the chosen sensor, some outlier measurements may appear in the obtained information.

3*) Heterogeneous landmark maps*: In some sector of the workspace the robot will see one map, and in another sector the map will be completely different. It must be taken into account to be able to position the robot among all these heterogeneous zones.

Several approaches are already used in start-up positioning. *Markov* methodology [2] uses a topological discretization of environment (grid), which covers a discrete set of robot feasible positions. It can easily be seen that an inadequate discretization of the workspace will result in an intractable grid. In *MonteCarlo* techniques [3], also called *particle filters*, the state is not discretized (i.e. they do not work with a grid); hence they require less memory resources. Another approach is the use of *neural networks*. In a paper from Hu and Gu [4], it is shown the result of using a *Kohonen Neural Network* in robot start-up positioning. However, neural networks suffer from lack of determinism, since the result obtained has always a percentage of confidence.

An interesting work made by Kieffer, Jaulin et al. [5], from CNRS, presents a methodology based in *interval analysis* techniques. In this approach they use a wide set of ultrasonic sensors placed on top of the robot, each one facing a different direction. This positioning is mainly static and slow, but very robust at the same time, allowing start-up without coded landmarks.

This paper shows how to apply interval analysis techniques for the start-up positioning of mobile robots equipped with a sensor that give bearing angles to known landmarks.

II. Focusing the Problem

In order to apply the proposed methodology, the robot must be equipped with any kind of sensor from which can be extracted bearing information. For instance, camera sensors can provide us with this information by means of applying computer vision algorithms. In this case, vertical edges as door frames or windows can act as landmarks. Another example of obtaining bearing angles is to use a laser scanner to detect landmarks.

In Fig. 1 we can see an example of the angles at which we would see some landmarks in our workspace.

Our problem is to find the [x,y,Ψ] triplet that has resulted in concrete sensor measurements: [θ_1, θ_2, … θ_n]. To simplify the problem, we suppose that the robot has not been moving until the first complete landmark map is obtained from the sensor (static start-up positioning).

[*] This research is sponsored by the Catalan Research Directorate, in the framework of its Reference Center of Advanced Production Technologies (CeRTAP).

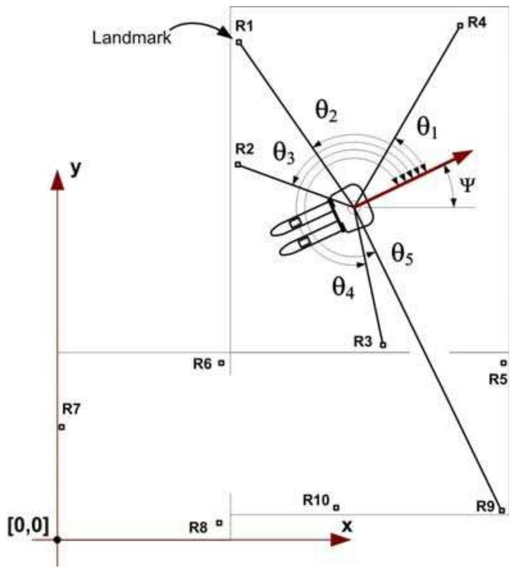

Fig. 1 Example of sensor measurements from a robot pose

III. Theoric Framework of Interval Analysis

Before we begin, we are going to give a view of the problem by using set theory mathematics. Our problem deals with the two sets shown in Fig. 2:

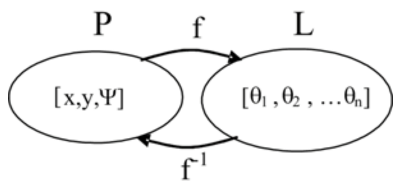

Fig. 2 The two sets implied in our problem

The set P contains every position-orientation triplet of the vehicle. The set L contains every possible n-tuple of goniometric measured angles. At last, we have a function called f that takes in an element of P, and using the knowledge of the workspace returns the corresponding element of L. Function f should be *injective*, otherwise there could exist ambiguities: it depends on the correct choose of landmarks and perhaps on other factors, but it falls outside the scope of this work.

However, it has not been found yet the inverse function f^{-1} that takes in a n-tuple of L and with the aid of workspace knowledge, may return the element of P, i.e. a $[x,y,\Psi]$ triplet. This paper presents an alternative way of finding f^{-1} by means of a technique based in interval analysis. The SIVIA algorithm (*Set Inversion Via Interval Analysis*) lets us solve set inversion problems: our case is a clear example, since we want to travel from a subset (interval) of L to the corresponding origin subset (interval) of P. This interval methodology allows us to take into account sensor imperfections and precision, so we finally get an interval of values for each angle in the n-tuple: in other words, a subset of elements of L.

Interval analysis techniques require the use of interval domains for all variables in the algorithm, even the Boolean ones. In addition to this, every function must be adapted to deal with intervals, as can be seen in [6].

We must also take into account the following aspects, specifics to our problem:

1) The elements of the two sets P and L will now be interval (P_i and L_i): $[\ [x_{inf}\ x_{sup}], [y_{inf}\ y_{sup}], [\Psi_{inf}\ \Psi_{sup}]\ ]$ and $[\ [\theta_{1inf}\ \theta_{1sup}], [\theta_{2inf}\ \theta_{2sup}], \dots [\theta n_{inf}\ \theta n_{sup}]\ ]$.

2) Function f, which went from P to L, now must go from P_i to L_i. So we must define a new function f_i which allows working with interval arithmetic.

IV. The SIVIA Algorithm

The *SIVIA* algorithm is as follows:

```
push([i_ini],S); // where S is a stack
while S ≠ ∅
   [i]=pop(S);
   if [f_test]([i])=[1 1] then push([i],S_S); // S_S: the “sure” result stack
   elseif [f_test]([i])=[0 0] then do nothing;
   elseif width([i])<ε then push([i],S_M); // S_M: the “maybe” result stack
   else {[i1],[i2]}=bisect([i]); push([i1],[i2],S)
end while;
return value = {S_S, S_M}
```

The functions that return intervals have been represented between square brackets [ ]. The aim is to compute the set [R] of elements of the origin that obey a property (represented by the function f_{test}, which evaluates true when that property is obeyed).

It begins dealing with an initial interval $[i_{ini}]$. While the algorithm loop is running, whenever the test function evaluates [0,1] the interval evaluated is divided into two halves (bisection). This way, all the search space will be classified by the test function. If doing a bisection, intervals are obtained that still evaluate [0,1], and their width is smaller than a predefined parameter ε, they are taken out of the search stack and passed to the “maybe” result stack. This shows that it has not been possible to define if the interval obeys or not the applied test function. This is the way of making the algorithm end in a finite time. The smaller the value of ε, the better the precision of the obtained result, but also the higher the CPU time required to finish.

If we call ΔR the union of intervals in the “maybe” result stack, and R^- the union of the “sure” result stack ones, then the desired set R can be enclosed the following way: $R^- \subseteq R \subseteq R^- \cup \Delta R$.

It has been said that SIVIA algorithm works with interval variables, but they are n-dimensional intervals, which can be called “*boxes.*” In our case, we will work with three-dimensional boxes (since the origin set called P was three-dimensional). When a bisection is made, the division is made across the wider dimension of the box.

In our positioning problem, the initialization is made by pushing 3 boxes in S stack. Each of them corresponds to each zone where the robot can physically be, as can be seen in Fig. 3:

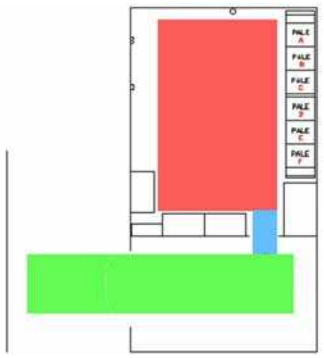

Fig. 3 The three boxes pushed in SIVIA stack initialization

In Fig. 3 above, the boxes are drawn in 2D, to help representing them in the workspace. But actually they are three-dimensional, with the third dimension varying among the interval [0, 2π] (i.e. orientation Ψ must also be obtained by the algorithm). It is shown in Fig. 4.

V. The Test Function

The way to adapt SIVIA to the start-up positioning problem is designing the f_{test} function specific to this problem, since the rest of the algorithm remains the same.

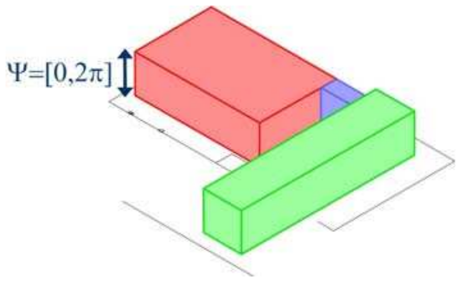

Fig. 4 SIVIA search boxes shown in 3D

The algorithm of our f_{test} is as follows:

$$land = \{\text{list of landmarks with their coordinates}\}$$
$$mea = \{\text{list of sensor measurements intervals, i.e. } [\theta\text{'s}]\}$$

$$int\_land = create\_list\_interval\_land(box, land)$$
$$[count] = \sum_{m \in mea} any\_int\_land\_in\_int\_m(int\_land, m)$$
$$f_{test} = \begin{cases} [1,1] \ , \ [count] = [\#_{mea}, \#_{mea}] \\ [0,0] \ , \ \text{upper\_bound}([count]) < (\#_{mea} - \#_{outliers\_allowed}) \\ [0,1] \ , \ \text{else} \end{cases}$$

As can be seen, the algorithm calls a function named *create_list_interval_land*, that computes a list whose elements are the intervals of angles from the present box to each landmark. The following figure shows the algorithm analyzing a box after having done several bisections. In Fig. 5 we can see how the function *create_list_interval_land* computes one of the intervals of angles mentioned before:

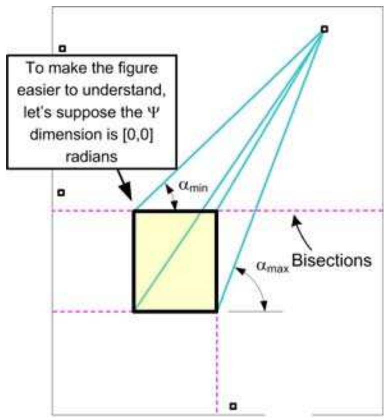
To make the figure easier to understand, let's suppose the Ψ dimension is [0,0] radians
α_{min}
α_{max}
Bisections

Fig. 5 Graphical view of obtaining a landmark's interval

The resulting interval for this landmark would be $[\alpha_{min}, \alpha_{max}]$. Remember that, in general case, the Ψ dimension of the box must be taken into account in the following way: $land\_interval = [\alpha_{min}, \alpha_{max}] + [\Psi_{min}, \Psi_{max}]$

Afterwards, the interval Boolean function *any_int_land_in_int_m* is computed for each measurement interval m, and the results are summed up into the interval variable [*count*]. The purpose of this function is to try to explain a real interval measurement from the hypothesis of the present box. For instance, we can see a measurement interval *partially explained* by a computed landmark interval in Fig. 6 below. In order to graphically represent the obtained landmark interval, the box is wrapped into a single point, from which all angle intervals come out.

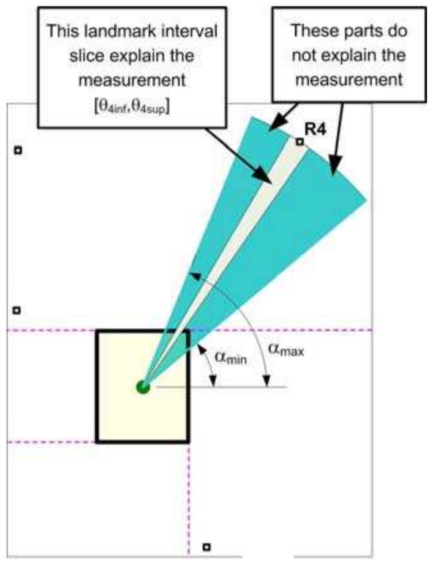
This landmark interval slice explain the measurement $[\theta_{4inf}, \theta_{4sup}]$
These parts do not explain the measurement
R4
α_{min}
α_{max}

Fig. 6 Graphical view of a landmark's interval partially explaining a goniometric measurement

The function result for that measurement interval in this situation would be [0,1], because there are some elements in the represented landmark interval that do not explain the measurement (i.e. that are not contained in the measurement interval).

At last, the return value of the test function is actually computed. If every measurement was explained by this box, the output is [1,1]. If it is not the case, the upper bound of [*count*] indicates indirectly how many measurements were not explained at all (i.e. how many returned [0,0] from *any_int_land_in_int_m*). If the number of unexplained is greater than the amount of allowed outliers, the test function returns [0,0]. Otherwise, it returns [0,1].

It must be noted that the correct way for dealing with outliers would be putting SIVIA inside a loop to be executed several times. The first run of the loop would begin with no outliers allowed, and as long as no result boxes are obtained at the end, it would be executed again with one more outlier allowed.

The test function presented works with no problem in rooms where all landmarks can be seen from any robot location in the room. Fig. 7 shows two robot locations with identical sensor measurement but pose 2 is unfeasible because landmark R5 can not be seen from that position. We need an extended test function that discards these unfeasible configurations taking into account the visibility of landmarks.

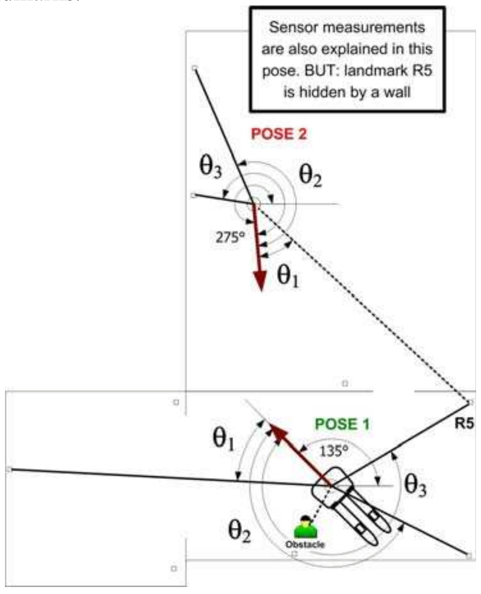
Sensor measurements are also explained in this pose. BUT: landmark R5 is hidden by a wall
POSE 2
θ3
θ2
275°
θ1
θ1
POSE 1
135°
R5
θ3
θ2
Obstacle

Fig. 7 Obstacle before reaching R5 makes unfeasible the pose 2

A. Extended Test Function

A little change in SIVIA algorithm structure will be introduced. A visibility vector will be associated to each box. This vector will be compound of interval Boolean values which will define visibility from the box to each reflector.

The SIVIA algorithm must be modified in order to always push both a box and its corresponding visibility vector. The first three boxes with which the algorithm is initialized will be associated with three visibility vectors with [0,1] values. The modified SIVIA algorithm is as follows:

```
push([$i_{ini}$],S); // where S is a stack
push(iVis$_{ini}$,V); // where V is a stack
while S $\neq \varnothing$
   [i]=pop(S);
   iVis=pop(V);
   [[out],iVis]=[fVis$_{test}$]([i],iVis);
   if [out]=[1 1] then push([i],$S_S$); // $S_S$: the “sure” result stack
   elseif [out]=[0 0] then do nothing;
   elseif width([i])<ε then push([i],$S_M$); // $S_M$: the “maybe” result stack
else {[i1],[i2]}=bisect([i]); push([i1],[i2],S); push(iVis,iVis,V);
end while;
return value = {$S_S$, $S_M$}
```

It can be seen that the test function is now called fVis$_{test}$. With this new situation, the test function is called with two arguments: the box and its visibility vector. And now it returns two output values: the interval Boolean result and the updated visibility vector. The new algorithm is as shown here:

$$land = \{\text{list of landmarks with their coordinates}\}$$

$$mea = \{\text{list of laser measurements intervals}\}$$

$$obst = \{\text{list of obstacles (mainly walls)}\}$$

$$vis\_out = \text{vector to be returned with updated visibility values}$$

$$vis\_out = vis\_in$$

$$\underset{r \in land \wedge vis\_out[r]=[0,1]}{\forall} \quad vis\_out[r] = recompute\_visibility(r, box, obst)$$

$$\underset{r \in land \wedge vis\_out[r]\neq[0,0]}{\forall} \quad int\_land[r] = create\_interval\_land(box, r)$$

$$[cnt] = \sum_{m \in mea} any\_vis\_int\_land\_in\_int\_m(int\_land, m, vis\_out)$$

$$f_{test} = \left\langle \begin{array}{l} \begin{cases} [1,1]\ ,\ [cnt] = [\#_{mea}, \#_{mea}] \\ [0,0]\ ,\ \text{upper\_bound}([cnt]) < (\#_{mea} - \#_{outliers\_allowed}) \\ [0,1]\ ,\ \text{else} \end{cases} \\ vis\_out \end{array} \right.$$

The first that the algorithm does is making a copy of the visibility vector argument into a temporal variable, called *vis_out*, which will be used for internal computations and returned at the end of the function as the updated visibility values. Afterwards, it re-computes the visibility from the present box to all those landmarks whose visibility value was [0,1]. This is due to the fact that when bisections are done and smaller boxes are generated, the [0,0] and [1,1] landmark visibilities will be inherited by the sub-boxes, so there is no need for reviewing them. However, if a landmark visibility was [0,1] in the parent box, it can now have derived to [0,0], [1,1] or remained the same [0,1]. An example of this situation is shown in the following Fig. 8. The parent box has [0,1] visibility for R9 landmark, so when its two child boxes are generated, visibility for this landmark must be recomputed for both. After obtaining new values, one of these two child boxes has [0,0] visibility for R9, so all of its future sub-boxes will inherit [0,0] value directly.

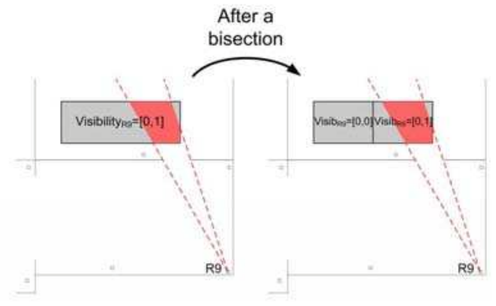

Fig. 8 Updating a visibility value after a bisection

Then, the list *int_land* is obtained as explained in the primitive test function. However, to avoid unnecessary processing time, it will now be computed only for landmarks with non-[0,0] visibility.

The function *any_vis_int_land_in_int_m* is used in a very similar manner as its counterpart in the primitive f_{test}. Now it's slightly modified to deal with visibility information.

$$\begin{array}{l} return \\ value \end{array} = \begin{cases} [1,1]\,, & \exists_{r\in land}\left(vis\_out[r]=[1,1]\wedge\left(int\_land[r]\in int\_m\right)=[1,1]\right) \\ [0,1]\,, & \neg\exists_{r\in land}\left(vis\_out[r]=[1,1]\wedge\left(int\_land[r]\in int\_m\right)=[1,1]\right) \\ & \quad\wedge\exists_{r\in land}\left(vis\_out[r]\wedge\left(int\_land[r]\in int\_m\right)\right)=[0,1] \\ [0,0]\,, & \text{else} \end{cases}$$

At last, the $fVis_{test}$ function assembles the tuple to be returned.

VI. Results

Our robot is an automated fork-lift truck. On the top of the robot there is a laser based goniometric sensor. It allows us to scan the working area by means of counter clockwise laser rotations. In the walls we have put several reflecting elements in known positions, acting as landmarks. Each time the laser detects any of them, we are warned of the detection and the angle at which it has occurred. We can see the vehicle and two reflectors in Fig. 9. Each time it passes by the 0 degrees angle (aligned with the robot), it sends us a signal, so we can know the exact angle at which we see every beacon with respect to the robot's own orientation. For the sake of simplicity, we will think of landmarks being points, i.e. without width. Actually, laser will give us a rising and a falling edge for each reflector, but only the rising edge will be taken into account.

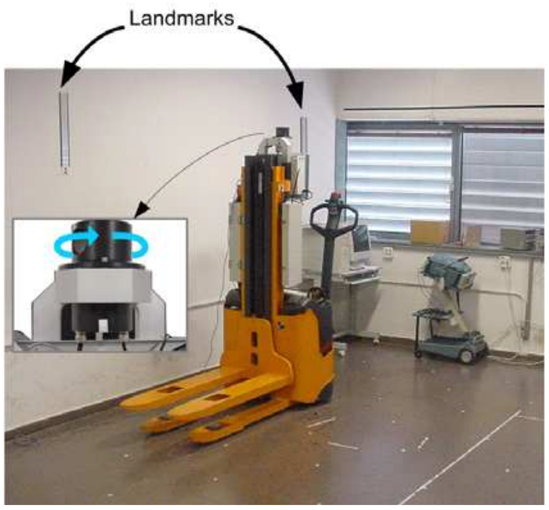

Fig. 9 Mobile robot in testing workspace

In order to see the response of the algorithm proposed in this paper at a realistic industrial environment, it has been tested at 14 different positions. Fig. 10 shows the robot at point P1, while the other 13 locations have been marked as circles. The coordinates of all tested positions are shown in Table I. It must be noted that for simplicity all the test poses have the same orientation: 0 rad. We have empirically determined that if the algorithm has been able to obtain the correct position of the robot, the found orientation will also be the correct one.

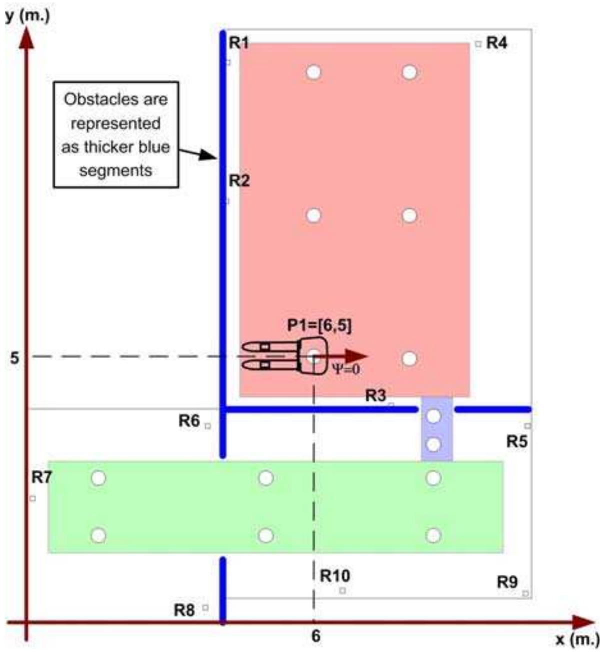

Fig. 10 Pose used for evaluating SIVIA algorithm

For each test point, the algorithm is executed many times, reducing the value of ε at each step, until it finds some [1 1] box. The obtained results are presented at Table I. Fig. 11 and Fig. 12 show an intermediate step and the final result, respectively, of the algorithm evaluation at P4 test point. It can be noticed that when the value of ε is 20 cm. the algorithm yields many scattered undetermined boxes (represented as yellow boxes). Afterwards, when the algorithm has progressed, and the value of ε has reached 0.3171 mm, not only there are less undetermined boxes but a number of [1 1] boxes (red) have appeared.

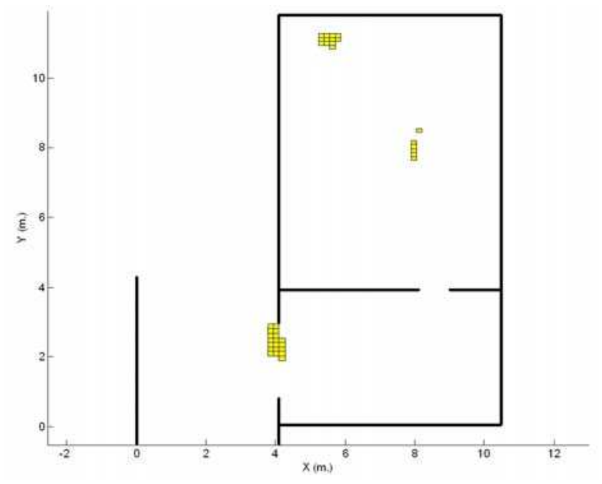

Fig. 11 Point P4, intermediate results at ε=0.20 m.

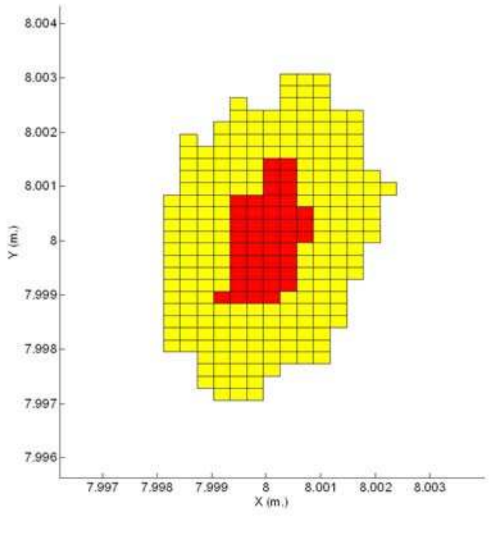

Fig. 12 Point P4, final step using ε=0.3171 mm.

Table I summarizes some relevant results of the execution of the algorithm over the 14 tested points. ε_{min} is the value of ε that has been needed for obtaining [1 1] boxes. $\mu_{[1,1]}$ is the mean of the centre points of all the [1 1] boxes found. The error distance between the actual test point location and the $\mu_{[1,1]}$ point obtained, is represented as d_{error}. At last, the execution time of the algorithm for each test point is represented as t_{exec}. It must be noted that the timing results have been obtained in a Pentium IV 3.2 GHz, using Matlab to implement the algorithm. The result boxes are shown in 2D.

TABLE I
TEST RESULTS

Point	Coord.	ε_{min} (mm.)	$\mu_{[1,1]}$ (m.)	d_{error} (mm.)	t_{exec} (s.)
P1	[6, 5]	0.75169	[6.0019,4.9984]	2.494	36.422
P2	[8, 5]	0.31712	[7.9997,5.0014]	1.426	16.824
P3	[6, 8]	0.31712	[5.9998,7.9999]	0.134	15.516
P4	[8, 8]	0.31712	[8.0000,8.0000]	0.074	15.891
P5	[6, 11]	0.31712	[5.9999,11.0000]	0.107	17.375
P6	[8, 11]	0.31712	[8.0001,11.0000]	0.128	16.641
P7	[8.5, 3.8]	0.31712	[8.4999,3.8000]	0.091	20.813
P8	[8.5, 3.2]	0.31712	[8.5001,3.2000]	0.107	15.156
P9	[1.5, 1.3]	0.31712	[1.4997,1.3000]	0.220	16.641
P10	[1.5, 2.5]	0.31712	[1.4999,2.4999]	0.076	20.516
P11	[5, 1.3]	0.31712	[5.0000,1.3000]	0.054	16.781
P12	[5, 2.5]	0.31712	[4.9998,2.5002]	0.248	16.406
P13	[8.5, 1.3]	0.31712	[8.5000,1.2999]	0.076	11.844
P14	[8.5, 2.5]	0.31712	[8.4999,2.5000]	0.050	14.422

As we can see in Table I, all the results give an error distance lower than 2.5 mm, which is more than good for start-up positioning in the industrial application of our fork-lift truck.

ACKNOWLEDGMENT

We wish to thank Luc Jaulin for his Interval Analysis Seminar that motivated us to use these techniques in our mobile robot start-up positioning.

REFERENCES

[1] C. Cohen, F.V. Koss, "A Comprehensive Study of Three Object Triangulation", *SPIE* Vol. 183 Mobile Robots VII (1992) / 95.
[2] J. Gutmann, W. Burgard, D. Fox, K. Konolige, "An experimental comparison of localization methods", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98), 1998.
[3] S. Thrun, D. Fox, W. Brugard, F. Dellaert, "Robust MonteCarlo localization for mobile robots", AAAI – 99.
[4] H. Hu, D. Gu, "Landmark-based navigation of autonomous robots in industry", IEEE International journal of industrial robot, vol. 27, no. 6, pp. 458 – 467, November 2002.
[5] M. Kieffer, L. Jaulin, E. Walter and D. Meizel, "Robust autonomous robot localization using interval analysis", *Reliable Computing*. **3**(6), pp. 337 – 361, 2000
[6] Moore, R.E., "Practical aspects of interval computation", Appl. Math., 13, 52-92