A Bachelor Thesis in Artificial Intelligence, University Of Amsterdam

Self-Localization in the RoboCup Soccer Standard Platform League with the use of a Dynamic Tree

Author:
Hessel van der Molen
hmolen@science.uva.nl
5619785

Supervisor:
dr. Arnoud Visser
A.Visser@science.uva.nl

Abstract

Locating (efficiently) a robot in an environment based on its observations is a hot topic in Robotics. There are several issues which have to be kept in mind like noise, computational power, ambiguity of observations et cetera. Based on an overview of current used localization methods in the RoboCup Soccer Standard Platform League, this thesis proposes a method which performs localization using a tree-like structure. The first part of the paper will show and compare in a short overview the current used methods and their issues and limitations. In the second part the proposed method is explained and justified. This is followed by test in re-allocation speed (when kidnapped) and accuracy, using the Gutmann data-set [7]. The results are compared with previous results of other methods tested with the used data-set, resulting in a short conclusion about the performance of the proposed method. Finally a discussion is presented which enumerates open issues and limitations which need further research.

keywords: Localization, RoboCup Soccer Standard Platform League, Particle Filer, Kalman Filter, Dynamic Tree, Gutmann data-set

June 24, 2011

Contents

1 Introduction

The year 2050,
Human vs. Humanoids,
The top of both worlds.

That's the ultimate goal of the RoboCup. In their way towards it, the RoboCup slowly grows and introduces different 'side' games like Robo Rescue and Dance. The Soccer League in its turn also got divided into several different leagues, each with its own set of rules. One of those league is the Standard Platform League (SPL) where each competing team competes with exact the same robots. Currently the Nao of Aldebaran is used[1].

In Robocup Soccer, things like soccer-field size and the color of the goals and lines are pre-determined. This makes it possible to do localization with an a priori known map. Moving at a field, a robot observes features. Features are specific points which identify a specific location. For example: a yellow pole can only be seen at the location of the yellow goal. In almost all localization algorithms such feature observations are described by the distance and angle to the feature seen from the robot-heading. Since such measurements are noisy (due to incorrect feature classification or incorrect distance or angle estimation) localization can become a difficult task.

As a thesis subject for the bachelor Artificial Intelligence of the University of Amsterdam, research towards localization in the SPL is done. Localization in this league is more interesting than other leagues because all teams have the exact same hardware configuration. Including the limited resources (memory and computational power), building or designing a localization-method which out-performs other methods and teams makes the challenge even more challenging.

The next sections will explain issues and limitations of current used systems, a description of the proposed method, test results and a conclusion with ideas for future research which can be done. A glossary of the used abbreviations is added after the "future research" section.

2 Related Work

In the RoboCup Soccer SPL 2011 are 28 teams competing. Most of the teams use different algorithms with different extensions with different limitations. In Table 1 a short overview is presented of the used methods together with the main pro's and con's[2]. A more detailed version can be found in Appendix A.

[1] http://www.aldebaran-robotics.com/

[2] As can be noted: the table does not contain all team-methods because some teams do not have a localization method while others do not describe it.

Used Method	Pro's	Con's	Teams using
(augmented) MCL	Proven to be accurate, Can handle kidnap problem, Can handle complex belief.	Comp. expensive!	Austrian-Kangaroos, B-Human, Cerberus, Edinferno, Noxious-Kouretes, TJArk
MCL & MOsr	Better results than MCL/sr		CMurfs
MCL & neg. Inf.	Faster elimination of particles than MCL		TT-UT Austin Villa
MCL & KF	Less comp. exp. than MCL		rUNSWift
AUX PF & SIR			SPQR+UChile
distance to goal poles	Simple	Not accurate	L3M, NTU RobotPal
UKF & MH	Smooth and performs well		Nao Devils Dortmund
muliple EKFs	Low computation cost		SPIteam
Constraint localization	Low computational cost, More adequate than PF		Nao Team Humboldt
Rao-Black & KF	Low computational cost Fast (re)localization		UPennalizers
Location Sensitive Behavior	Simple	Reliability issues	Wrighteagle Unleashed!
Local Model	Simple	No communication between the robots	WPI Warriors
Cox &CI & UKF	High potential	Not yet stable	RoboEireann

Table 1: Short overview of used methods in SPL 2011

As can be seen from Table 1, most teams are using a Particle Filter approach (Austrian-Kangaroos, B-Human, Cerberus, Edinferno, Noxious-Kouretes, TJArk, CMurfs, TT-UT Austin Villa, rUNSWift and SPQR+UChile). Based on these findings it could be concluded that a Particle Filter would be the best option for localization. It is also proven that such a method can result in accurate localization. This however comes at a cost: it is more computational expensive, compared to other methods. There are some extensions developed to decrease the complexity, but still, there is the question if a less computational method could result in similar performance.

Another thing which is not well handled in a basic PF is kidnapping (putting a robot at a different location when is tries to estimated its current position). This problem is removed with the use of, for example, random re-sampling of the state space [13]. The same problem does occur in methods based on a (dynamic) set of Kalman Filters (Nao Devils Dortmund, SPIteam and UPennalizers). The source of poor response towards kidnapping is because these methods sample the state space. Placing the robot at a location were no samples describe the state space results in poor (or incorrect) pose-estimation. Random re-sampling makes sure that even when the system has converged to a specific location, the rest of the state space is described, and thus, theoretical, can re-locate the

robot at a new position. A better solution would be to incorporate all possible states. Such an solution requires however a fundamental different approach. A method which has such a fundamental different approach is Constraint Localization (used by Team Humboldt), which estimate a position based on the overlapping space of regions in which certain features at certain distances can be found.

Besides kidnapping and sampling of the data, feature ambiguity can also be a problem: how is it possible to determine at which feature the robot is looking when there are more of such features in the environment? One solution is to only incorporate unambiguous data, like detecting only a yellow or blue goal (done by L3M and NTU RobotPal). The problem with such an approach is that much (potential valuable) data is ignored, resulting in a pose estimation which is likely to contain larger errors compared to methods including all data. Solving such an problem can be done by calculating the relative distance between two features and the distance towards the robot, or by estimating the distance to all individual features, and calculating the intersection between these distances. The first system requires that two features are seen in the same observation, while the latter ignores this information. The best solution for this would be to combine both systems.

3 Proposed Method

To drop the need of state re-sampling, a different way the represent a belief needs to be created. The proposed method does not model the belief as an exact position of the robot, but it creates a region with a probability that the robot is in that region. When an environment is divided into small regions, each having the same size, Grid-Localization [20] would be the method to use. This method however has a large drawback: to estimate the position of a robot in an accurate way, the map needs to be divided into small regions, resulting in the need of much memory. An unwanted effect of such a representation is, is that regions having a low probability are maintained with the same accuracy as regions with a high probability.

A solution for this problem would be to recursively divide the map into smaller regions (in essence, creating a kd-tree). If the probability of a certain region (a parent) is above a threshold (or if the region meets a set of rules), the region is expanded (creating children). When the probability gets below an another threshold (or if the region meets a different set of rules), the children of that region are removed. In this way a position of a robot can be estimated with high accuracy, while at the same time memory costs stay low. The rules to expand or collapse can be adjusted easily to incorporate values of neighbors, parents, grandparents et cetera. In theory, this method should be easily extendable.

Other (assumed) advantages of such a Dynamic Tree Localization (DTL) method, are:

- A complex belief can be easily represented: two children of the same parent can be expanded at the same time.

- The online computation cost can be reduced by building the tree-structure in advance: nodes can be set on/off to determine if it is used or not[3].
- Convergence to the correct location can go quick: when using a k-nary Dynamic Tree of depth d, the node at depth d is $\frac{1}{k^d}$ of the size of the root-node.
- Large lists of past observations are not needed: all information is processed in the probability of a node in the tree.
- Representing the complete state-space with probabilities makes the system able to handle noisy data and kidnapping.

The next sub-sections will describe the method in a more detailed way.

3.1 Description of the Method

The method is build around a tree-like structure. Each node in this tree represents a specific area or region of the field. Throughout this thesis such a node will be called a 'block' or 'node'. In Algorithm 1 to 4 the pseudo-code of the DTL method is shown.

Algorithm 1 Main structure of the Dynamic Tree Algorithm

```
tree = CreateRootAndRootChildren()
loop
  observation = GetObservation()
  tree = UpdateTree(tree, observation)
  tree = CheckCollapse(tree)
  tree = CheckExpand(tree, maxTreeDepth)
end loop
```

Algorithm 2 UpdateTree(tree, observation)

```
for all obs ∈ observation do
  for all node ∈ tree do
    if IsIn(obs, node.range) then
      p = ProbObsInNode(obs, node)
      node += p
      sibling(node).prob -= (p / number_of_siblings)
      node.heading = UpdateHeading(obs, node)
    end if
  end for
end for
return tree
```

The main program only consists of four basic functions: retrieving a list of observed features, updating the probabilities of the tree with these observations

[3]This will however consume more memory than a grid-localization approach

Algorithm 3 CheckCollapse(tree)

```
for all node ∈ tree do
  if node.prob ∪ COLLAPSE_RULES then
    node.RemoveChildrenTree()
  end if
end for
return tree
```

Algorithm 4 CheckExpand(tree)

```
for all node ∈ tree do
  if node.prob ∪ EXPAND_RULES then
    children(node) = MakeChildren(node, NUM_OF_CHILDREN)
    for all child ∈ children(node) do
      child.range = CalcDistAngleRange(field, child)
      child.prob = CalcChildProb(node)
    end for
  end if
end for
return tree
```

and collapsing or expanding the tree.

Like almost all localization methods, the list of observed features contains the signature of the feature, the estimated distance towards it and the angle seen from the heading of the robot. All observed features are propagated through all nodes in the current belief (the tree). Using the distance and heading of each feature the probability of a block is updated (Algorithm 2). An update consists of calculating the probability that a feature can be seen at the given distance from a block, scaling and adding this to the probability of that block and decreasing the probability of the block-siblings. Adding and subtracting the probability makes it possible to update the blocks a-synchronous: it does not matter which block is updated first. Updating in such a way also requires that the calculation of *ProbObsInNode(obs, node)* does not include the current probability. After updating (increasing) the probability of a node, the probability of the siblings of the node is decreased to keep the total probability of all siblings to 1:

$$1 = P(node_1|parent(node_1), parent(parent(node_1))...root)+$$
$$P(node_2|parent(node_2), parent(parent(node_2))...root)+$$
$$... + P(node_n|parent(node_n), parent(parent(node_n))...root)$$

With the use of such a structure, universal collapse- and expand- rules can be made (they do not have to adjust for example thresholds to compensate the depth of a node).

To estimate the complete pose of a robot, each block contains an estimate of the current heading of the robot.

Once all observations are propagated through the tree, sets of rules determine if a block needs to be expanded or collapsed. Collapsing a block results in the removal of all the (grand)children which entails that the accuracy of the area described by the block (and its children) decreases. Expanding works the other way around. When a block is expanded it has to be ensured that the total of the surface described by the children equals the surface described by the block itself (otherwise the belief is corrupt: parts of potential states are missing):

$$surface(child_1) \oplus surface(child_2) \oplus ... \oplus surface(child_n) = surface(parent)$$

If the children are created, a list with all minimum and maximum distances towards all features in the environment is created. Having such a list makes it easy to check if a feature is observable from a certain block, which increases the (online) speed of the method. Besides a distance range, also the angle range is calculated, seen from a so-called 0-heading (the angle towards the point where the heading of the robot is set to be 0) . This range can be used to estimate the complete pose of the robot.

4 Implementation

The complete system is programmed with Python 2.6.1[4]. For pose-estimation OpenCv 2.1 was used and the GUI is build using Pygame 1.9.1. The next couple of sections will describe the implementations of the routines used for DTL.

4.1 Field, Observation and Orientation representation

The field is represented as a list with signatures of features. Features which looks the same, but are placed at different locations have the same signature. This makes it possible to create only a few "chunks" which contain all detectable features in a field. Per signature a list is made which contains the coordinates of all features with that signature:

$$Sig_1 = [[x_1, y_1], [x_2, y_2], [x_3, y_3], ..., [x_n, y_n]]$$

$$Sig_2 = [[x_1, y_1], [x_2, y_2], [x_3, y_3], ..., [x_m, y_m]]$$

$$...$$

$$Sig_k = [[x_1, y_1], [x_2, y_2], [x_3, y_3], ..., [x_l, y_l]]$$

These lists are then combine into one list, called "Field", where each index of "Field" represents the signature of a feature:

$$Field = [Sig_1, Sig_2, ..., Sig_k]$$

The x and y positions of a feature are calculated with the origin at the bottom-left corner. The position along the horizontal axis (width of the field) is described by x, while the position along the vertical axis (height) is described by y. The heading of a robot is 0 when it looks in the vertical direction (the

[4]Python 2.6.1 is used because the current installed naoqi (1.10.44) is using this. Using a newer Python version would result in the risk of creating programs which can not be run on the Nao

0-heading). An angle left of this direction is represented with a value between $-\pi$ and 0. An angle right of this direction is described with a value between 0 and π.

The representation of an observation (e.g. a snapshot with a camera) is also an list, which has the following format:

$$Observation = [Sig_1, R_1, \theta_1, Sig_2, R_2, \theta_2, ..., Sig_n, R_n, \theta_n]$$

Were n is the number of features found in a observation, R is the distance between the robot and the observed feature, θ the angle between the feature and the current heading of the robot and Sig is the signature of the feature. The used dataset is first transformed into such a structure, before it is fed to the algorithm.

4.2 Tree Representation and Creation

With no clue of how fast node-expanding online would be, the complete tree is made in advance. Therefore each node-structure contains more (and different) information than a node-structure for online computation would have. The used structure looks like:

```
Structure: Block
    parent: <Block-Object>
    child: [<Block-Object>, <Block-Object>, ...]
    probability: <0-1>
    childActive: <0 (nonactive) or 1 (active) >
    centerlocation: [x,y]
    heading: [\theta, [obs1, obs2, ... , obsN]]
    depth: <0-MaxDepth>
    ID: <To discrinate between nodes>
    featurelist: <same as Field-Representation, but now [x,y]
                  of each feauture is changed in
                  [R_{low}, R_{up}, \theta_{low}, \theta_{up}] >
```

The *parent* and *child* variables points to respectively the parent of the node and all the children (the latter is a list of pointers). The *childActive* variable holds the information if the children of the node are currently incorporated in the belief of the robot. By default all blocks are set non-active (beside the root-block and its children) with an even distributed probability. Estimation of the heading is done by keeping track of the last n observations which are assumed to be seen from this block. The first value in this list is the estimation, the other values are the last n observations (in the programmed system, this value is set to 5)[5]. The estimate is calculated by weight-averaging the observations: the last observation gets the highest weight, the first observation the lowest (from 1/3 to 0)[6]. Determining if a node is at the maximum depth is done using the *depth* variable. Calculation the ranges of all features in *featurelist* is done using Figure 1 and Table 2. A derivation and a more detailed description of the rules can be found in Appendix B.

[5]Seemed to be a nice number: other values could also be used

[6]This is also a 'feel-good' - value: no tests are done to adjust (improve) it

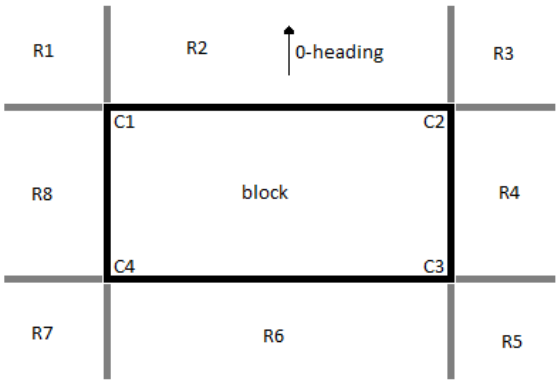

Figure 1: Regions around a Block in which a feature can be

Table 2: Rules for determining the correct ranges

Region	Distance Range	note:	Angle Range	max. range (degrees)
1	$\|c1 \to F\| < distance < \|c3 \to F\|$	-	$c2 < angle < c4$	$-90 < angle < 0$
2	$\|top \to F\| < distance < \|c3 \to F\|$	$F < middle$	$c2 < angle < c1$	$-90 < angle < +90$
2	$\|top \to F\| < distance < \|c4 \to F\|$	$F > middle$	$c2 < angle < c1$	$-90 < angle < +90$
3	$\|c2 \to F\| < distance < \|c4 \to F\|$	-	$c3 < angle < c1$	$0 < angle < +90$
4	$\|left \to F\| < distance < \|c1 \to F\|$	$F < middle$	$c3 < angle < c2$	$0 < angle < +180$
4	$\|left \to F\| < distance < \|c4 \to F\|$	$F > middle$	$c3 < angle < c2$	$0 < angle < +180$
5	$\|c3 \to F\| < distance < \|c1 \to F\|$	-	$c4 < angle < c2$	$+90 < angle < +180$
6	$\|bottom \to F\| < distance < \|c2 \to F\|$	$F < middle$	$c4 < angle < c3$	$+90 < angle < -90$
6	$\|bottom \to F\| < distance < \|c1 \to F\|$	$F > middle$	$c4 < angle < c3$	$+90 < angle < -90$
7	$\|c4 \to F\| < distance < \|c2 \to F\|$	-	$c1 < angle < c3$	$-180 < angle < -90$
8	$\|right \to F\| < distance < \|c2 \to F\|$	$F < middle$	$c1 < angle < c4$	$-180 < angle < 0$
8	$\|right \to F\| < distance < \|c3 \to F\|$	$F > middle$	$c1 < angle < c4$	$-180 < angle < 0$

4.3 Probability Estimation

Besides the tree-structure, the probability estimation for a block plays a major role in the whole system. The used calculation is not mathematical justified, it is only a pure intuitive implementation which, with experimenting and adjusting some values, seemed to work.

The basic idea to calculate the probability is to include several things:
(o = feature observation, b = block, p = all parents&grandparents)

- $p(o|b)$
- $p(o|p)$
- $p(o)$

With Bayes rule, $p(b|o)$ can be calculated, which after scaling could be added to the current $p(b|\forall o)$.

$p(o|b)$
This probability is calculated by creating a Gaussian:

$$p(o|b) = min_p + (max_p - min_p) * e^{\frac{(d-c)^2}{2*(1/4*w)^2}}$$

In this formula, c is center of the distance-range of a block to a observed feature, w is the total width of the range and d is the observed distance towards the feature. The idea behind this setup is that if the measured distance is close to the the edge of the block, it is more probable that the measurement does not belong to the assumed block, but to its neighbor. The resulting Gaussian is scaled to a probability of 1 (max_p), if the distance is the distance to the center of the block and roughly 0.1 (min_p) if it is at the end of the range.

$p(b|p)$
Since all probabilities of the children sums up to 1, the probability $p(b|p)$ can be transformed to $1/NumberOf(parent(node).children))$.

$p(o)$
It can be reasoned that the accuracy of a distance-estimation at a long distance is more inaccurate than the measurement of a distance close-by. To model this, the relation between the inaccuracy of the measured distance is assumed to be linear: if the distance is twice as far, the inaccuracy is twice as big. To define the boundaries of the output, the maximum measurable distance (from corner to corner of the soccer-field) is first calculated. This distance is assumed to have a probability of 0.1. Distances between 0.0 and 1.0 meter are assumed to have a probability of 0.9 (no distance-calculation is perfect). All values in-between are linear scaled.

Putting the above described calculations together with Bayes rule, $p(b|o)$ can be calculated:

$$p(b|o) = \frac{p(o|b)(p(b|p)}{p(o)}$$

Since this output is to big to add to the current probability of a the block, it is divided by 15. This value is experimentally determined (just like all above described values).

4.4 Expanding and Collapsing

The rules for expanding and collapsing the tree are set to the bare minimum. A block is collapsed if the probability is below a threshold, independent of values of other blocks. Expanding is done when the probability is above a different threshold, also independent of other blocks. This set-up is chosen to be able to make a stronger claim about the potential of the method: the more naive and basic the method is implemented, the more potential a method has if it performs accurate or acceptable.

After expanding a block, the calculation of the probability of the new (child) blocks are done by first setting all probabilities equal, and then propagating the

last 5 overall observations[7] (not the last observations used for heading estimation). This is done to get a faster convergence and a lower massive-expanding rate if the expand threshold is below $1/num\_of\_childs$. The heading estimation of the children-block is set equal to the heading estimation of the parent. The heading-observations are not included. Setting this estimate overcomes the problem of a temporally estimation of an initial value (since the heading *needs* a value).

5 Testing

The next couple of sections describe the way of testing the proposed method and what data-set is used.

5.1 The used dataset

For a dataset the Gutmann-dataset is used [7]. The data-set is a 58 minute recording of a walking AIBO ERS 2100 (equipped with a CMOS camera). The robot walked multiple times an 8-figure in a 3x2m environment. In this environment 6 different color beacons were placed for recognition. The robot was joy-sticked around 5 different mark points. Each time the robot passed a mark-point, a tag was added to the dataset. Besides the distance and angle towards the color-beacons, the odometry estimation of the robot got also recorded. Gutmann et. al. also made modified versions of this dataset to test on accuracy, re-location (kidnapping), noise handling and data sparseness.

Besides all datasets, Gutmann et. al. also included their best path-estimation. This estimation is used to test kidnap-performance. There is however a small inconsistence between several time-steps in the reference-log and the observation-logs. To be able to use the datasets to test the DTL method, all inconsistent data is removed (from all files). Besides removing the inconsistent data, also all measurements which do not include a feature observation are removed. This is done because the implemented system does not incorporate such data in any way. In total 8197 (from 51667 lines to 43470) records of the used observation-log are deleted and 4110 records (from 25948 to 21838) from the kidnap-log.

5.2 Comparing estimates

Because DTL does not return a pose-estimation, and the reference log does, a filter is build which takes in the tree and returns the most likely pose. For this calculation the field is represented as an image, in which the centers of all final nodes are printed. The value of a center is calculated with the block probability ($p(b|field)$ and not $p(b|p)$) multiplied by the inversed block size with respect to the field size:
(k = number of children per block, d = depth of block)

$$center = p(b|field) * k^d$$

After printing all final-node centers, the image is smoothed, followed by a search to the pixel with the highest value. It is assumed that this pixel represents the

[7] Just like many other values: determined by doing a few simple tests

current location of the robot.

To compute the heading direction, the heading-estimates of all blocks of which the center lies within a certain distance from the estimated robot position are averaged. Each heading is averaged with the same weight.

5.3 Used parameters and tests

The method is tested on its accuracy and re-location time, using the script and utilities provided by Gutmann et. al. Since the dataset does not contain a ground truth, the accuracy of the method is tested based on its distance error towards each mark. This error is not the 'true' error of the method: walking the robot at the mark position in not 100 percent accurate. It is also possible that there are small time-delays in writing the marks to the log-file. These errors are not an issue, since the tests Gutmann et. al. performed also included these errors. Testing the re-location time is done by comparing the markers, the reference log, and the resulting log of DTL.

All tests are done using a binary tree of different depths. The accuracy of the pose-estimation image is set to 1 pixel per cm. Smoothing is done with a 25 pixel (cm) wide 2D Gaussian with a standard deviation of 10. The distance between the positions estimate and a block center for calculating the heading of the pose is set to 10cm.

Determining the best values for the tree depth (4, 6, 8, 10 or12) and the expand (0.4 to 0.85 in steps of 0.05) and collapse threshold (0.1 to 0.6 in steps of 0.1) is done by brute force. The best parameters are then used to test the re-location time needed when the robot is kidnapped.

6 Results

In Figure 2 the results are printed (with a 95% confidence) of the accuracy tests (in Appendix C the tables of these results can be found).

As expected, the accuracy of all tree increases when the collapse and expand thresholds are increased. Increasing the expand threshold makes it harder to split block: the robot has to be more certain it is in that block before the accuracy is increased. If the collapse-threshold is set too high, the system removes too quick too much data, therefore the system cannot converge towards an accurate position. Remarkable is that the best accuracy distances of all trees are close to each other, while the block-size is quite different (as can be seen in Table 3).

The distance error of the DTL implementation is roughly twice as big as the best implementation of Gutmann et. al. Their absolute distance errors lay between 87 and 122 mm.

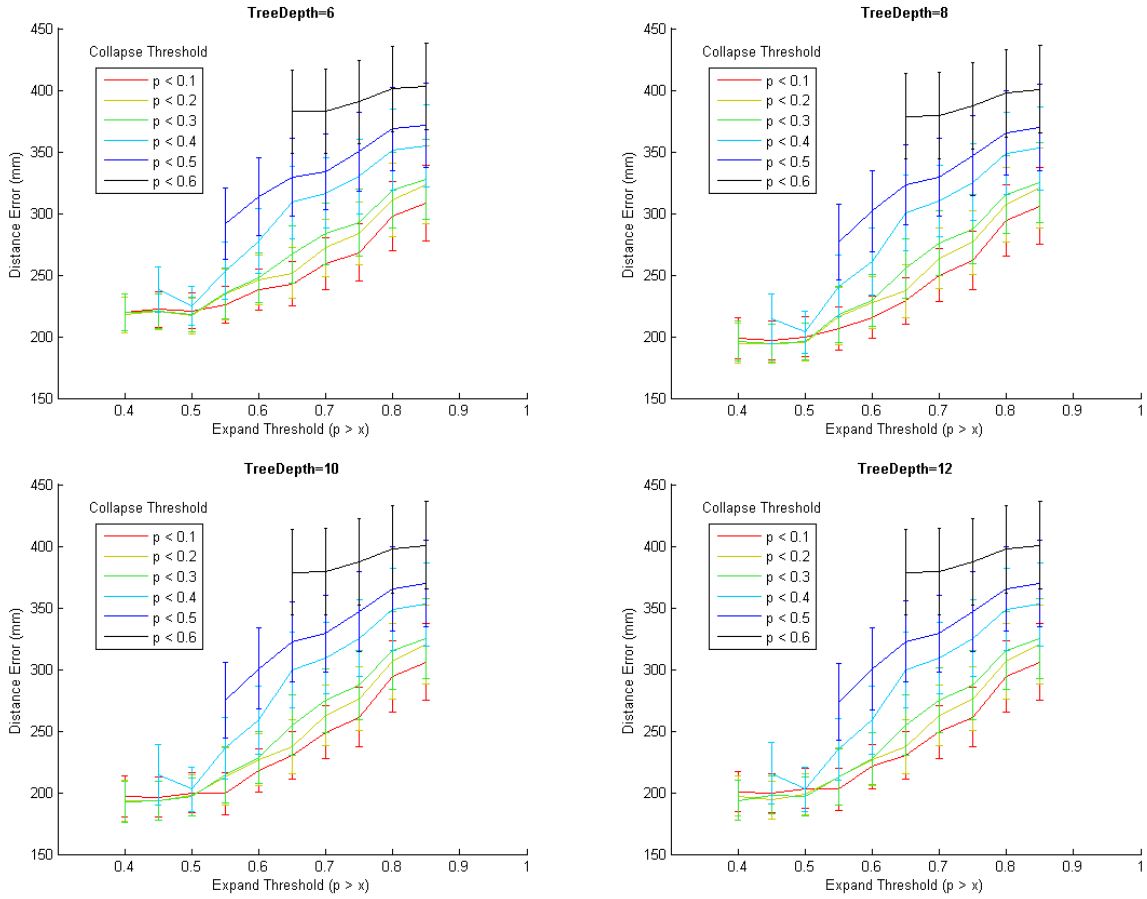

Figure 2: Accuracy-test results with different tree-depths and expand/collapse conditions.

tree depth	block size	best absolute error
6	187 x 250 mm	220 mm
8	94 x 125 mm	194 mm
10	47 x 63 mm	193 mm
12	23 x 31 mm	193 mm

Table 3: Smallest block sizes at different tree depths

From the graphs, it is also possible to read that there are no 'best'-parameters: the smallest absolute errors lay all between each other confidence bounds. Nonetheless, for kidnapping the parameters with the smallest absolute errors are chosen: $expanding > 0.45$ and $collapsing < 0.2$.

The results for re-location after kidnapping can be found in Table 4.

tree depth	time taken	95% confidence
6	4.0 sec	0.9 sec
8	4.1 sec	0.8 sec
10	4.8 sec	1.2 sec

Table 4: Time needed for relocation Re-location using: $exp > 0.45$, $col < 0.2$

The decrease of the needed time to re-locate when using a smaller tree can be assigned to the idea that a smaller tree needs less observations to expand fully and a new location. The implemented system only expand or collapse nodes after an observation update. So the minimum observations needed to re-locate at maximum accuracy, is for a tree with depth 6, 6 observations, while it is for a tree of depth 8, 8 observations. This, because in used rules a root-node may collapse while its children are expanded.

Comparing the results of kidnapping in DTL with the results of the tests performed with Gutmann et. al., DTL can be placed in the middle range of the tested methods. (see Figure 3)

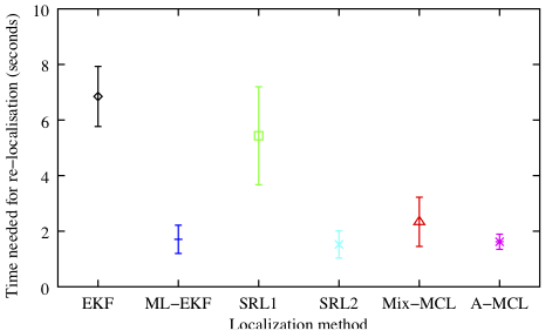

Figure 3: Results of tests performed by Gutmann et. al. (Reprint of Fig. 7 from [7]).

7 Discussion and Conclusion

Based on the current used methods in the Robocup Soccer Standard Platform League and their issues and limitations a different localization approach is presented. For this approach a kd-tree is adapted resulting in: Dynamic Tree Localization. DTL has the advantages that it incorporates all possible states in the state space (no 'fix' for kidnapping needed). It can also represent a complex belief, is robust against noise and has a fast convergence towards small regions. Tests on accuracy and kidnap-recovery show that DTL can be placed between current state of the art techniques: localization is done with an accuracy of 20 cm and re-location after kidnapping is done in 4 seconds. The results are brute-force created by testing several parameters: de maximum dept of the tree, the expansion threshold and the collapse threshold. The smallest absolute error is returned with a tree of a maximum depth of 10 (representing block with a size of 47 x 63 mm) and an expansion threshold of $p > 0.45$ and a collapse thresh-

old of $p < 0.2$. Since these tests are done with a basic ad-hoc implementation, extensions can be easily made to further increase the accuracy and performance.

The used data-set does not include solid information about the heading a the robot. Therefore it is hard to validate the implemented method for heading-estimation. Watching the GUI, it looks like the estimation algorithm estimates the heading quite well.

8 Future Research

As mentioned above, some extensions can be made to further increase the accuracy of DTL.

The implemented probability calculation is quite ad hoc. A look at this calculation could have a positive effect on the current performance. The implemented system does also not incorporate a smoothness function at the edge of block. If a feature is observed at the edge of one block, just out of the range of a neighbors block, the neighbors probability should also be updated since the measurement is noisy and the feature could belong to this neighbors block.

The sets of rules to expand and collapse can also be improved by adding more rules. It is likely to occur that a parent (representing a large region) has a low probability, while one of its grand children (representing a small region at the edge of a parent) has high probability, which is correct. Including information about parents or siblings to determine if a node needs to expand can also result in faster expanding (lower threshold) when needed, while blocks with exact the same probability are not expanded.

In this thesis, tests are only done with a binary tree. When more children are used, the size of a block at a certain depth does decrease much faster. If this decreasing results in better (faster) localization and re-localization should be tested.

An another extensions would be to incorporate in the range-calculations of a block, a special feature: the no-observation feature. Seeing no feature also tells the robot something about its heading and position.

Above all, the most valuable test would probably be to test the method on a robot. Current tests are only done off-line with a data-set.

9 Acknowledgments

Special thanks to my supervisor, Arnoud Visser, for guiding me in testing and setting up the idea for Dynamic Tree Localization. Also thanks to the Dutch Nao Team for testing and implementing the method on their system.

10 Glossary

DTL	Dynamic Tree Localization
SPL	Standard Platform League
KF	Kalman Filter
UKF	Unscented Kalman Filter
EKF	Extended Kalman Filter
MCL	Monte Carlo Localization
PF	Particle Filter
AUX PF	Auxiliary Particle Filter
SIR	Sample Importance Re-sampling
MH	Multi Hypothesis
Rao-Black	Rao-Blackwellized
Cox	Cox algorithm
CI	Coveriance Intersection

References

[1] A. Burchardt, T. Laue, and T. Röfer. Optimizing particle filter parameters for self-localization. *RoboCup 2010: Robot Soccer World Cup XIV*, pages 145–156, 2011.

[2] F. Caron, M. Davy, E. Duflos, and P. Vanheeghe. Particle filtering for multisensor data fusion with switching observation models: Application to land vehicle positioning. *Signal Processing, IEEE Transactions on*, 55(6):2703–2719, 2007.

[3] B. Coltin and M. Veloso. Multi-observation sensor resetting localization with ambiguous landmarks. In *Proceedings of AAAI*, 2011.

[4] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte-carlo localization:efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 343 – 349, 1999.

[5] D. Göhring, H. Mellmann, and H.D. Burkhard. Constraint based belief modeling. *RoboCup 2008: Robot Soccer World Cup XII*, pages 73–84, 2009.

[6] D. Gohring, H. Mellmann, and H.D. Burkhard. Constraint based world modeling in mobile robotics. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2538–2543. IEEE, 2009.

[7] J.-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 454 – 459 vol.1, 2002.

[8] T. Hester and P. Stone. Negative information and line observations for monte carlo localization. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2764–2769. IEEE, 2008.

[9] J. Hoffman, M. Spranger, D. Gohring, and M. Jungel. Making use of what you don't see: negative information in markov localization. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2947–2952. IEEE, 2005.

[10] J. Hoffmann, M. Spranger, D. Göhring, and M. Jüngel. Exploiting the unexpected: Negative evidence modeling and proprioceptive motion modeling for improved markov localization. *RoboCup 2005: Robot Soccer World Cup IX*, pages 24–35, 2006.

[11] K. Kaplan, B. Celik, T. Meriçli, C. Meriçli, and H. Akın. Practical extensions to vision-based monte carlo localization methods for robot soccer domain. *RoboCup 2005: Robot Soccer World Cup IX*, pages 624–631, 2006.

[12] T. Laue and T. Röfer. Particle filter-based state estimation in a competitive and uncertain environment. In *Proceedings of the 6th International Workshop on Embedded Systems. VAMK, University of Applied Sciences; Vaasa, Finland*. Citeseer, 2007.

[13] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 1225 –1232 vol.2, 2000.

[14] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the National conference on Artificial Intelligence*, pages 593–598. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[15] N. Özkucur and H. Akın. Localization with non-unique landmark observations. *RoboCup 2010: Robot Soccer World Cup XIV*, pages 72–81, 2011.

[16] M. Quinlan and R. Middleton. Multiple model kalman filters: A localization technique for robocup soccer. *RoboCup 2009: Robot Soccer World Cup XIII*, pages 276–287, 2010.

[17] M.J. Quinlan, SP Nicklin, N. Henderson, R. Fisher, F. Knorn, SK Chalup, RH Middleton, and R. King. The 2006 nubots team report. *School of Electrical Engineering & Computer Science Technical Report, The University of Newcastle, Australia*, 2007.

[18] T. Röfer, T. Laue, and D. Thomas. Particle-filter-based self-localization using landmarks and directed lines. *RoboCup 2005: Robot Soccer World Cup IX*, pages 608–615, 2006.

[19] P.L. Rosin and G.A.W. West. Nonparametric segmentation of curves into various representations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(12):1140 –1153, dec 1995.

[20] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. Intelligent robotics and autonomous agents. MIT Press, 2005.

[21] S. Mcdonald J. Whelan, T. Stüdli and R. Middleton. Line point registration: A technique for enhancing robot localization in a soccer environment. In *Proceedings RoboCup Symposium*, pages 2538–2543, 2011.

A An Overview of used Localization Methods in the RoboCup Soccer Standard Platform League 2011

A.1 Teams and their localization

A.1.1 Austrian-Kangaroos

URL: *http://www.austrian-kangaroos.com/*
TQP 2011: *http://www.austrian-kangaroos.com/downloads/tqp2011_austrian-kangaroos.pdf*
TDP 2010: *https://sites.google.com/a/austrian-kangaroos.com/public_en /download/tdp2010_austrian-kangaroos.pdf?attredirects=0&d=1*

In their 2011 Team Qualification Documen, the Austrian Kangaroos state that they will improve their current used localization method, which (as described in the Team Description Paper of 2010) is an implementation of Monte-Carlo Localization as described in [20]. Further they state that they will use a line detection method as described in [19].

A.1.2 B-Human

URL: *http://www.b-human.de/*
Code Release 2010: *http://www.b-human.de/file_download/33 /bhuman10_coderelease.pdf*

As described in their 2010 Code-Release, B-Human uses a Monte Carlo Localization method as described in [4]. This, as described in [18] is proven to provide accurate results in the RoboCup Soccer Standard Platform League and can deal with the kidnapped robot problem. The exact implementation is the augmented MCL version [7].

The used filter, uses (during an update) only 6 randomly chosen perceptions received from its sensors. These perceptions include: goal posts (ambiguous as well as unambiguous ones), line segments (of which only the endpoints are matched to the Field model), line crossings (of three different types: L, T, and X), and the center circle. To increase reliability, B-human also introduces a Pose-Validation method. This method computes a translation and rotation based on the current sensor input (goals, center circle and seen lines). With the use of a two-dimensional Kalman Filter the translation is filtered (using current sensor input and MCL output) and with the use of one-dimensional Kalman filter the rotation is filtered (also using current sensor input and MCL output).

For ball tracking (velocity and position) a particle filter is used [12]. The particle filter first estimates both values. Then, twelve different Kalman filters determine the exact values. From these twelve filters, only one is chosen to be the "most true" (based on the covariance -smaller is better- and how well the'seen' position matches the estimated position). The Kalman filters are further divided into two sets: one for rolling balls and one for stationary balls. After each frame-update the worst distribution of a set is replaced by a new initialization.

When a ball is stopped by a robot, the Kalman Filter with the best probability (most-true) distribution is 'clipped' onto the robot's feet. The new calculated

values for velocity and position are calculated (and depending) from the motion of the feet.

In [1], B-Human shows some calculated Parameters for Self-Localization for a Particle Filter.

A.1.3 Burst

URL: *http://shwarma.cs.biu.ac.il/robocup*
-

A.1.4 Cerberus

URL: *http://robot.cmpe.boun.edu.tr/∼cerberus/wiki/*
TDP 2011: *http://robot.cmpe.boun.edu.tr/∼cerberus/wiki/uploads/Downloads/Cerberus2011-TDP.pdf*

Cerberus uses also MCL. The have extended their MCL method with 'practical extensions' described in [11]. Their implementation uses unidentified observations (inspired by FastSLAM [14] and Multi-Hypothesis tracking [20]) resulting in a: Switching Observation Model [2]. Observed objects which look the same (but could be different objects) are places in similar groups (identified with the same identifier). For example: a T-type of field line intersection can be found at 6 different locations in a field. All these locations are labeled with the same identifier. At a location multiple identifiers from different groups can be seen, which makes a certain location more probable than other locations. More about this method can be found in [15].

A.1.5 CMurfs

URL: -
TDP 2011: *recieved from email*

In combination with Multi-Observation sensor resetting [3], CMurfs is using a MCL implentation. Besides updates by odometry and observations (lines and goals), MCL is also updated when the state of the game change (penalty, start) or when the state of the robot does change (falling over). The Multi-Observation sensor resetting extention is added because converges more efficiently to the true robot position and it is less sensitive to noise than standard sensor restting.

For ball localization multiple hypothesis from all robots are combined and maintained. The hypothesis with the highest probability is chosen to be true.

A.1.6 Dutch Nao Team

URL: *http://www.dutchnaoteam.nl/*

Localization method is described in this thesis.

A.1.7 Edinferno

URL: *http://www.ipab.inf.ed.ac.uk/robocup/*
TDP 2010: *http://www.ipab.inf.ed.ac.uk/robocup/pubs/EdinfernoTDDoc.pdf*

MCL is also used by Edinferno. The system detects in real time goal locations, line intersections and the inner circle. The filter implemented is the augmented version [20] to avoid problems such as kidnapping and false convergence. For a motion model, odometry information is used.

A.1.8 L3M

URL: *http://www.lisv.uvsq.fr/~hugel/robocup/robocup.htm/*
Team Report 2011: *http://www.lisv.uvsq.fr/~hugel/robocup/qualification/2010/2010_SPL_L3M_Team_Report.pdf*
TQP 2011: *http://www.lisv.uvsq.fr/~hugel/robocup/qualification/2011/research_description.pdf*

Currently, the localization system of L3M is only based on the goal-poles: the lines on the field are not yet supported/recognized. The system simply calculates the distance and angle between the two blue or yellow poles and estimates the position of the robot on these values. This however results in poor localization when the poles can not be seen or are obstructed by other robots. For improvement the idea is to use the green borders or the green seen polygon of the field and map this to a top view to use correlation techniques for a rough estimate. The white lines are used for near localization such a for positioning the goal keeper or a attacker in the middle of the field.

A.1.9 Mi-Pal

URL: *http://www.mipal.net.au/*
-

A.1.10 MRL-SPL

URL: *?*
-

A.1.11 Nao Devils Dortmund

URL: *http://www.nao-devils.de/*
Team Report 2011: *http://www.irf.tu-dortmund.de/nao-devils/download/2010/TeamReport-2010-NaoDevilsDortmund.pdf*

For localization, the Nao Devils Dortmund use line intersections (L, T and X-type), the inner circle and both goals. The used method is a combination of a Unscented Kalman Filter and a Multi-Hypothesis system [16]. This combination is chosen because of it's "smoothness and performance". The Multi-Hypothesis system is needed to recover from large localization errors due position changes (kidnapping / bumping onto other robots / falling over) or errors in odometry without correction provided by observation. The inaccuracy of the odometry (when the robot does "fast biped walking") amplifies the error-problems.

For filter updating they use the most likely data association: each observed feature is combined with a probability (unique things - like, a goal - gets a higher probability). Creating such a data association is in itself also a problem: how

is determined which data is best? For this, a solution is described in [16]]: each Gaussian is split en several new ones, each representing different association choices, which are applied to all hypothesis. The Nao Devils came up with a more efficient (in memory complexity) way:

They only generate a few new hypotheses at positions with a high probability, based on recent sensor information. The new generated hypothesis are only updated when the data lies inside a certain (expected) threshold. In combination with some other approximations and simplification they are able to create a method which is an order of magnitude faster than the -by them- previously used particle filter, while providing superior localization quality and increased robustness to false positive perceptions.

A.1.12 Nao-Team HTWK

URL: *http://robocup.imn.htwk-leipzig.de/*
-

A.1.13 Nao Team Humboldt/

URL: *http://www.naoth.de*
Team Report 2009: *http://www.naoteamhumboldt.de/wp-content/uploads/2010/02/NaoTH09Report_final.pdf*
TDP 2010: *http://www.naoteamhumboldt.de/wp-content/uploads/2010/02/NaoTH10Description.pdf*

In the 2009th team report, team Humboldt state that they are using a Kalman filter for local ball localization. Global locations of the ball are communicated between the Nao's. Constraint localization is used for self-localization.

This team state that in earlier years a particle filter performed adequate for self localization, however due to the increased complexity of the localization (colored flag removal, increasing field size) the number of particles necessary gets to a computational limit. Using to low particle numbers, complex belief can't be represented anymore. Therefore Team Humboldt started using localization based on constraints [5] [6].

Perception is taken from flags, goals and lines. The shape of the constraint depends on the kind of data (a line percept result in is rectangle shaped constraint, a flag or goal in a circle/ellipse) and noise (depending on percept distance). Handling all resulting constraints is done as follow (quote is from their 2010 TDP):

> After having generated all constraints, we propagate the constraints with each other as long as there are no more constraints or the resulting solution space becomes empty. The position belief of the robot is stored in form of constraints as well and propagated with the sensory constraints as well. If, for some steps, the belief doesn't fit to the sensory constraints, or even if no new sensory data are available, the belief constraint boarders are increased at first. If sensory data remain inconsistent, we reset the belief to the sensor data constraints.

A.1.14 Northern Bites

URL: *http://robocup.bowdoin.edu/blog/*
-

A.1.15 Noxious-Kouretes

URL: *http://www.kouretes.gr/*
TQP 2011: *http://www.intelligence.tuc.gr/kouretes/docs/2011-noxious-kouretes-application.pdf*

The Noxioud-Kouretes are using MCL. Their belief is represented by (x, y, θ) and updating the belief is done by an auxiliary particle filter. For the motion model, odometry of the robot is used (exhibiting all locomotion) and for a sensor model a landmark model for both goals is used. Re-sampling of the particles is done by a linear time re-sampler with including a selection with replacement implementation. The robot's pose is chosen to be the particle with the highest weight.

A.1.16 NTU RobotPal

URL: *http://www.csie.ntu.edu.tw/ bobwang/RoboCupSPL/index.html*
TDP 2009: *http://www.csie.ntu.edu.tw/ bobwang/RoboCupSPL/ NTU_Robot_PAL_09Report.pdf*

In their 2009 team description paper , NTU RobotPal described that they wanted to use a MCL implementation. Due to insufficient time, they, in 2009, used a much simpler method: tracking the goal positions and calculating the localization with respect to the goals.

A.1.17 NUbots

URL: *http://robots.newcastle.edu.au/*
-

A.1.18 Portuguese Team

URL: *http://www.ieeta.pt/portugueseteam*
-

A.1.19 RoboEireann

URL: *http://www.eeng.nuim.ie/robocup/*

RoboEireann works on a slightly new approch[8]: a variant of Cox's algorithm together with Covariance Intersection and an Uncented Kalman Filter [21]. The performance of this method (as described in [21]) is so-far not optimal, but performs equal or better then by-them tested localization methods.

[8]This information is gained from e-mail

A.1.20 rUNSWift

URL: *http://cgi.cse.unsw.edu.au/∼robocup/2010site/*
Team Report 2010: *http://www.cse.unsw.edu.au/∼robocup/2010site/reports/report2010.pdf*

The combination of a Monte Carlo Particle Filter and a Uni-Modal Kalman Filter is used by rUNSWift. In the initial position of the robot the particle filter is used to converge the belief to a single location. Once converged, this location is the starting point of the Kalman filter. They maintain a so-called *kidnap factor* which is the discrepancy between the observations and the Kalman filter's state. If this values exceeds a certain threshold, the more computational expensive particle filter is used to converge again. In this way, they take the best of both methods. In their team-report they explain more details about their implementation.

A.1.21 SPIteam

URL: *http://www.spiteam.org/*
TQP 2011: *http://www.teamchaos.es/documents/SPIteam-qualification-2011.pdf*

The SPIteam uses multiple EKFs for localization. They have chosen this method over a particle filter due to computational costs. The idea to overcome problems such as kidnapping is to use and maintain a dynamic population of EKFs, each representing the belief about a particular position of the robot. This population is maintained and varied in several ways:

- New EKFs are created to describe a position were the robot could be, but which is not yet described by an EKF;
- EKFs which have proven to be credible are removed from the population;
- When different EKFs describe similar positions, they are combined.

SPIteam claims that this method is"extensively tested and the results show how it is a reliable, robust and computationally inexpensive method to calculate the robot position at the RoboCup Field.

A.1.22 SPQR+UChile

URL: *?*
TDP 2011: *https://www.tzi.de/spl/pub/Website/Teams2011/spqruchile_TDP.pdf*

SPQR is using an Auxiliary Variable Particle Filter and Sample Importance Re-sampling. They further have integrated information/hints when to use sensor resetting and when to uses which method. This results in a system which picks the most suitable strategy/parameters for a specific situation. UChile has improved localization by estimating independently and in addition to the robots pose, the pose of static and mobile objects. This makes it possible to use dynamic landmarks (temporally and spatially local objects). They state that this

is especially valuable when attention-demanding tasks (e.g. ball tracking) are performed and that (using this feature) the robot is able to correct its odometry even when it is lost (which goes into the direction of human-like ways of localization).

note: some how there are no references in the TDP while they do refer to articles.

A.1.23 TeamNanyang

URL: *http://sites.google.com/site/teamnanyang*
-

A.1.24 TJArk

URL: *http://see.tongji.edu.cn/TJArk/English/contact.html*
TQP 2011: *http://see.tongji.edu.cn/TJArk/download/papers/TJArkNaoApp2011.pdf*

TJArk uses -just like many others- MCL for localization. The used motion model uses the calculated odometry of the robot. For the observation model all objects perceived (goals, center circle, lines and intersections) are used. Ball tracking is also done with a MCL implementation. Tracking information is shared with the team mates if a Nao knows quite certain where the ball is.

A.1.25 TT-UT Austin Villa

URL: *http://www.cs.utexas.edu/∼AustinVilla/?p=naol*
TDP 2010: *http://www.cs.utexas.edu/∼stone/Papers/bib2html-links/UTAITR1101-spl10.pdf*

Again, MCL is used for localization. In their TDP they also show some enhancements they introduced in MCL: incorporating negative information and line information [8]. Negative information is for example the expectation to have a certain observation, but not having that observation. If this happens, it is likely to be the case that the robot is not in the position it thinks it is. Using this information, particles can be eliminated even when no landmarks are observed. More about this method can be found in [10] [9].

Ball tracking and localization is done using a 4-state Kalman filter and with respect to the robot. This method is based on the 7-state Kalman filter presented in [17].

TT-UT Austin Villa further keeps track of 3 opponents. If an observed opponent is within three meters of a previous detected opponent, the global location of that robot is updated, otherwise it is replaced by the observed one. The system also assumes that - when kicking a ball - all robots stay at their current location for 6 seconds.

A.1.26 UPennalizers

URL: *http://fling.seas.upenn.edu/∼robocup/wiki/*
Team Report 2010: *http://www.seas.upenn.edu/∼robocup/files/*

upennalizers_team_research_report_2010.pdf

The UPennalizers state that "complex probabilistic models can be difficult to implement in real-time due to lack of processing power on board of the robots". They address this problem with an algorithm that "incorporates a hybrid Rao-Blackwellized representation that reduces computational time while still providing high level of accuracy". The algorithm models the uncertainty of the pose as a distribution over a set of heading angles (discrete) and translational coordinates (continuous). The heading angle is described/updated with discrete Markov updates while the translation of the robot is updated with a Kalman Filter. When implemented, the system results in quick localization, even after a robot is kidnapped and replaced.

A.1.27 WPI Warriors

URL: *http://users.wpi.edu/∼soniac/RoboCup/*

The WPI Warriors don't have a localization method yet[9]. Instead they create a local world model based on their vision system, which worked quite well: they were able to win two games at the US open 2011. However, they hope to have a localization method at the World Cup, but which is yet to determine.

A.1.28 Wrighteagle Unleashed!

URL: *http://wrighteagleunleashed.org/l*
Team Report: *http://wrighteagleunleashed.org/*

Wrighteagle Unleashed! have experimented with non traditional approaches, resulting in a system of location-sensitive behaviors. This system works according three principles:

- for each observed object the robot calculates the position of the object relative the its torso (using the altitude of the object -ball is lies on the ground-);
- By default, the robot walks in the direction of the ball;
- Simultaneously the robot tries to line up for a kick towards the goal.

Having designed "reliable and straight kicks" this system is quite effective. It has however some weaknesses: e.g. if the robot is near/inside the goal area it has some reliability issues.

[9]This information is gatherd via email

A.2 Summary

Of all 28 teams, 19 descriptions for localization are found. These can be summarized as follow:

Used Method	Pro's	Con's	Teams using
(augmented) MCL	Proven to be accurate, Can handle kidnap problem, Can handle complex belief.	Comp. expensive!	Austrian-Kangaroos, B-Human, Cerberus, Edinferno, Noxious-Kouretes, TJArk
MCL & MOsr	Better results than MCL/sr		CMurfs
MCL & neg. Inf.	Faster elimination of particles than MCL		TT-UT Austin Villa
MCL & KF	Less comp. exp. than MCL		rUNSWift
AUX PF & SIR			SPQR+UChile
distance to goal poles	Simple	Not accurate	L3M, NTU RobotPal
UKF & MH	Smooth and performs well		Nao Devils Dortmund
muliple EKFs	Low computation cost		SPIteam
Constraint localization	Low computational cost, More adequate than PF		Nao Team Humboldt
Rao-Black & KF	Low computational cost Fast (re)localization		UPennalizers
Location Sensitive Behavior	Simple	reliability issues	Wrighteagle Unleashed!
Local Model	Simple	No communication between the robots	WPI Warriors
Cox &CI & UKF	high potential	not yet stable	RoboEireann

Table 5: Short overview of used methods in SPL 2011

B Derivation: Determining Distance and Angel range towards a feature, seen from a Block

All space around a block in a field is divided into 8 different regions, as can be seen in Figure 4 (R1-R8). This is done to make it more easily to determine the angle-range to see a feature, given the 0-heading.

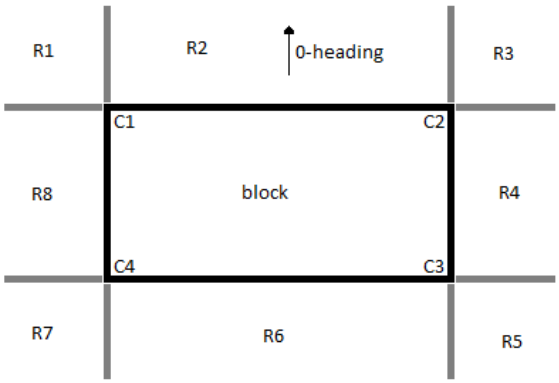

Figure 4: Imaginary regions around a block

B.1 Distances

This sections uses Figure 5.

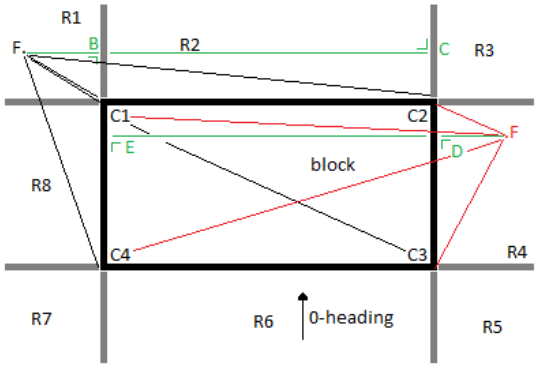

Figure 5: Distances from features to corners/sides of a block

B.1.1 The range of region 1, 3, 5 and 7

Assume that there is a feature F in Region 1. It is now possible to draw a straight line between F and a point B and C. These drawn lines are perpendicular to the 0-heading. From Pythagoras we know that in a triangle yields: $c^2 = a^2+b^2$, where a and b are perpendicular to each other. Using this, $|C1 \to F|$ can be described as:

$$|C1 \to F|^2 = |B \to C1|^2 + |B \to F|^2$$

From the figure, it is easy to see that $|F \rightarrow C| > |F \rightarrow B|$ and $|B \rightarrow C4 > |B \rightarrow C1|$, therefore the distance between F and $C1$ is the shortest from all possible points int the block.

Determining the largest distance is done using the same comparison ($|F \rightarrow C| > |F \rightarrow B|$ and $|B \rightarrow C4 > |B \rightarrow C1|$) and the equation: $|B \rightarrow C4| = |C \rightarrow C3|$ and $|B \rightarrow C| = |C4 \rightarrow C3|$ it can be concluded that C3 is the furthest away from the seen feature in Region 1.

Since all corner-regions are in some way a mirrored version of each other, this proof can used do determine the other ranges. This results into:

Region	Distance-range
1	$\|c1 \rightarrow F\| < distance < \|c3 \rightarrow F\|$
3	$\|c2 \rightarrow F\| < distance < \|c4 \rightarrow F\|$
5	$\|c3 \rightarrow F\| < distance < \|c1 \rightarrow F\|$
7	$\|c4 \rightarrow F\| < distance < \|c2 \rightarrow F\|$

B.1.2 The range of region 2, 4, 6 and 8

For this proof, a feature is assumed to be in Region 4.

Again, using Pythagoras, it can be seen that the distance $|D \rightarrow F|$ is the shortest of all, independent where F is in R4. Computing the farest distance is however depending on the location in R4.

If F is above the middle of R4 ($|E \rightarrow C1| < |E \rightarrow C4|$), the distance $|C4 \rightarrow F|$ is the largest of all (Pythagoras..).If the feature is however below the middle of R4, the largest distance is $|C1 \rightarrow F|$. Again, this reasoning can be applied to all non-corner regions, which results into:

Region	Distance-range	note
2	$\|top \rightarrow F\| < distance < \|c3 \rightarrow F\|$	$F < middle$
2	$\|top \rightarrow F\| < distance < \|c4 \rightarrow F\|$	$F > middle$
4	$\|left \rightarrow F\| < distance < \|c1 \rightarrow F\|$	$F < middle$
4	$\|left \rightarrow F\| < distance < \|c4 \rightarrow F\|$	$F > middle$
6	$\|bottom \rightarrow F\| < distance < \|c2 \rightarrow F\|$	$F < middle$
6	$\|bottom \rightarrow F\| < distance < \|c1 \rightarrow F\|$	$F > middle$
8	$\|right \rightarrow F\| < distance < \|c2 \rightarrow F\|$	$F < middle$
8	$\|right \rightarrow F\| < distance < \|c3 \rightarrow F\|$	$F > middle$

B.2 Angles

B.2.1 The range of region 1, 3, 5 and 7

Since all corners are in some way the mirrored version of each other, only a proof is give for one region.

First a arbitrary feature in region 1 is assumed. Then, a triangle can be drawn between $C1$, $C2$ and F (see Figure 6, left image). Because the two gray lines pointing towards the 0-heading are parallel to each other, the angle $c2$ can

be re-written at the gray line on the left. Now is can be easily determined that c2 is always bigger than c1:

$$c1 + . + *^{10} = 180 = c2 + .^{11} \Rightarrow c2 = c1 + * \Rightarrow c2 > c1$$

Because the relation between $C4$ and $C3$ equals the relation between $C1$ and $C2$, but only lowerd, it is justified to state that $c3 > c4$.

Determining if $c1$ is bigger than $c4$ (or $c2$ bigger than $C3$) is easily done, the same method as above can be applied (see Figure 6, right image):

$$c4 + . + *^{12} = 180 = c1 + .^{13} \Rightarrow c1 = c4 + * \Rightarrow c1 > c4$$

Having proven that $c2 > c1 > c4$ and $c3 > c2$, it can now be stated that the angle range to see F in region 1 is described by: $[c4, c2]$, where $c4 < observation\_angle < c2$.

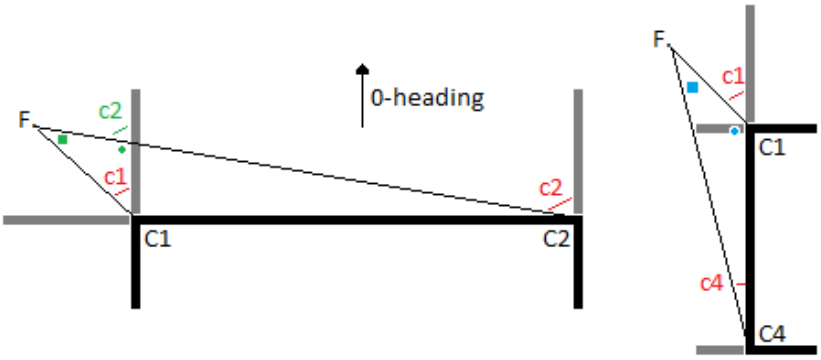

Figure 6: Traingles desrcibed by the corners of a block and a feature

In the used application, all angles at left of the 0-heading are described as being negative (from 0 to -180 degrees). Therefore the angle-range to see feature F has to be rewritten as: $[c2, c4]$, where $c2 < observation\_angle < c4$ (with a maximum range of: $-90 < observation\_angle < 0degrees$).

A feature in region 3 inherits the same reasoning as a feature in region 1, only the angles are now positive: ranging from 0 to +180 degrees. The angle-range for region 3 is thus described with: $[c3, c1]$, where $c3 < observation\_angle < c1$ (with a maximum range of: $0 < observation\_angle < +90degrees$).

Reflecting the desribed ideas to region 7 creates an output which is described by the range $[c3, c1]$. Just like a feature in region 1, the angles are negative, changing the range into $[c1, c3]$, where $c1 < observation\_angle < c3$ (with a maximum range of: $-180 < observation\_angle < -90degrees$).

[10] sum of corners in a triangle
[11] corner of a straight line
[12] sum of corners in a triangle
[13] corner of a straight line

Table 6: Rules for determining the correct ranges

Region	Distance Range	note:	Angle Range	max. range (degrees)
1	$\|c1 \rightarrow F\| < distance < \|c3 \rightarrow F\|$	-	$c2 < angle < c4$	$-90 < angle < 0$
2	$\|top \rightarrow F\| < distance < \|c3 \rightarrow F\|$	$F < middle$	$c2 < angle < c1$	$-90 < angle < +90$
2	$\|top \rightarrow F\| < distance < \|c4 \rightarrow F\|$	$F > middle$	$c2 < angle < c1$	$-90 < angle < +90$
3	$\|c2 \rightarrow F\| < distance < \|c4 \rightarrow F\|$	-	$c3 < angle < c1$	$0 < angle < +90$
4	$\|left \rightarrow F\| < distance < \|c1 \rightarrow F\|$	$F < middle$	$c3 < angle < c2$	$0 < angle < +180$
4	$\|left \rightarrow F\| < distance < \|c4 \rightarrow F\|$	$F > middle$	$c3 < angle < c2$	$0 < angle < +180$
5	$\|c3 \rightarrow F\| < distance < \|c1 \rightarrow F\|$	-	$c4 < angle < c2$	$+90 < angle < +180$
6	$\|bottom \rightarrow F\| < distance < \|c2 \rightarrow F\|$	$F < middle$	$c4 < angle < c3$	$+90 < angle < -90$
6	$\|bottom \rightarrow F\| < distance < \|c1 \rightarrow F\|$	$F > middle$	$c4 < angle < c3$	$+90 < angle < -90$
7	$\|c4 \rightarrow F\| < distance < \|c2 \rightarrow F\|$	-	$c1 < angle < c3$	$-180 < angle < -90$
8	$\|right \rightarrow F\| < distance < \|c2 \rightarrow F\|$	$F < middle$	$c1 < angle < c4$	$-180 < angle < 0$
8	$\|right \rightarrow F\| < distance < \|c3 \rightarrow F\|$	$F > middle$	$c1 < angle < c4$	$-180 < angle < 0$

The last corner-region to desribe, is region 5. Just as region 3 is the positive-angle version of region 1, region 5 is the postive-angle version of region 7. The range is thus described by: $[c4, c2]$, where $c4 < observation\_angle < c2$ (with a maximum range of: $90 < observation\_angle < 180 degrees$).

B.2.2 The range of region 2, 4, 6 and 8

From the proof of the previous section (Figure 6, right image) it is known that (for region 2) $c1$ is always bigger than $c4$ and $c2$ always bigger than $c3$. Since angles left of the 0-heading are defined as negative, and the maximum range has to be found, $c1$ and $c2$ are used for defining the range for region 2. In region 2, $c2$ has always a negative angle, therefore the range is described as: $[c2, c1]$, where $c2 < observation\_angle < c1$ (with a maximum range of: $-90 < observation\_angle < +90 degrees$).

Region 6 is the mirrored version of region 2, so, the range of region 6 is described with: $[c4, c3]$, where $c4 < observation\_angle < c3$ (with a maximum range of: $+90 < observation\_angle < -90 degrees$). This means that the angle should be bigger than 90 degrees (between 90 and 180) or smaller than -90 degrees (between -90 and -180). The resulting construction is however something which has to be kept in mind during programming.

Region 8 and 4 are rotations of 90 degrees of region 2. This lets the range easily be described. For region 8, the range is: $[c1, c4]$ where $c1 < observation\_angle < c4$ (with a maximum range of: $-180 < observation\_angle < 0 degrees$). Region 4 is described by: $[c3, c2]$ where $c3 < observation\_angle < c2$ (with a maximum range of: $0 < observation\_angle < +180 degrees$)

B.3 Conclusion

With angles left the the 0-heading described as -180 to 0, and angles at the right of the heading with 0 to 180, the angle and distance range for a feature in a region is described by:

C Accuracy Results

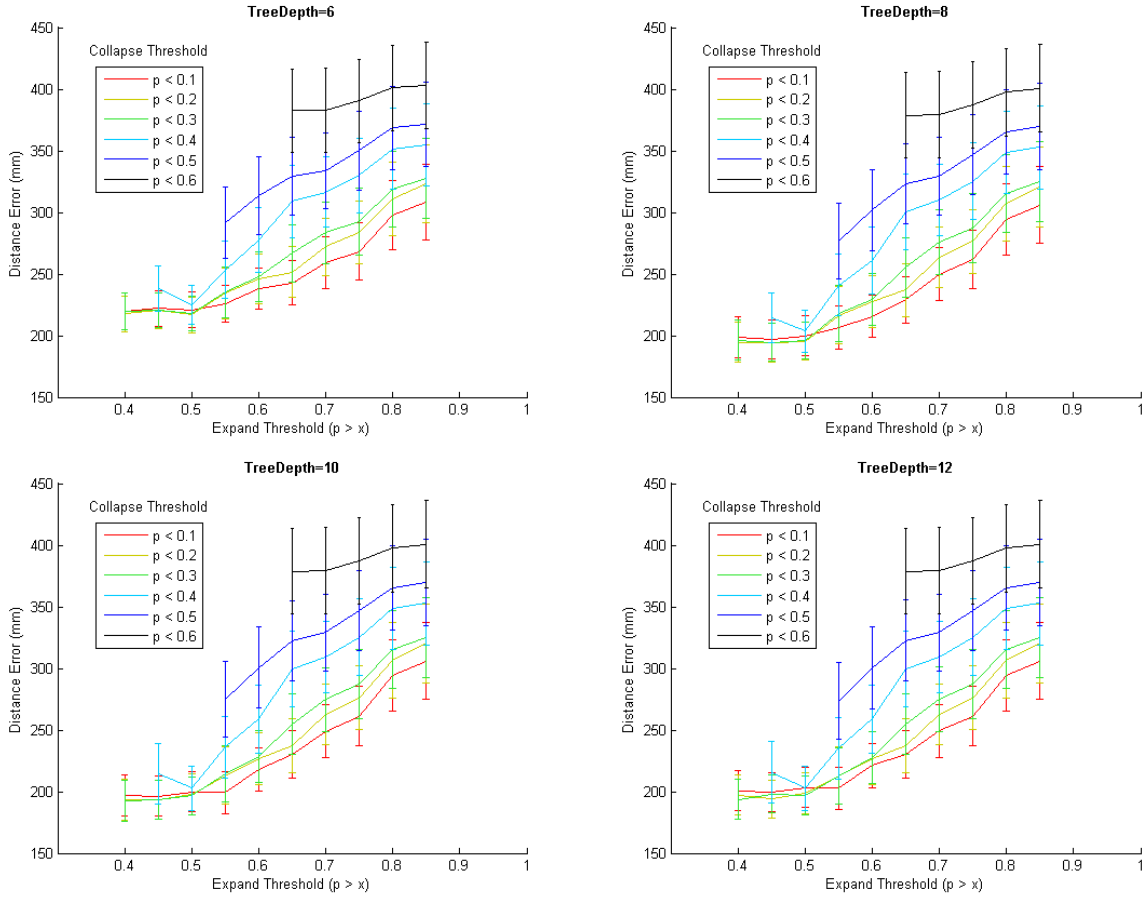

Figure 7: Accuracy-test results with different tree-depths and expand/collapse conditions.

The above shown figures are created with the Tables on the next page (Table 7a to Table 7d. These results show the accuracy (average error) of multiple trees with different maximum depths and different parameters for collapsing and expanding the tree. As can be seen the accuracy of all trees is around 20mm. Increasing the thresholds for expanding and collapsing also results in an increase of the inaccuracy. The confidence bounds are set of 95 percent.

(a) Distance error (in mm) and 95% confidence (in mm) of a tree with a depth of 6

col \exp	$p > 0.4$	$p > 0.45$	$p > 0.5$	$p > 0.55$	$p > 0.6$	$p > 0.65$	$p > 0.7$	$p > 0.75$	$p > 0.8$	$p > 0.85$
$p < 0.1$	219.8 / 14.8	222.2 / 14.4	221.0 / 14.8	225.6 / 14.7	238.0 / 16.4	242.8 / 17.8	259.0 / 21.0	268.2 / 23.2	297.9 / 28.3	308.0 / 30.7
$p < 0.2$	217.8 / 14.5	220.1 / 14.2	216.8 / 14.2	234.2 / 20.1	246.2 / 19.9	251.6 / 20.8	272.0 / 23.5	283.4 / 25.4	311.1 / 29.9	323.3 / 31.9
$p < 0.3$	219.7 / 14.6	220.8 / 14.0	218.0 / 14.1	235.3 / 20.8	248.0 / 20.3	266.8 / 23.3	283.3 / 25.2	292.9 / 27.2	318.9 / 31.0	327.7 / 32.1
$p < 0.4$	x	238.2 / 18.3	224.8 / 15.5	253.2 / 23.3	277.5 / 26.6	308.8 / 29.6	316.7 / 28.1	329.9 / 30.5	351.7 / 32.8	354.9 / 33.4
$p < 0.5$	x	x	x	291.5 / 29.0	313.5 / 31.7	330.0 / 31.6	333.9 / 30.5	350.2 / 32.0	368.5 / 33.8	371.6 / 34.3
$p < 0.6$	x	x	x	x	x	382.5 / 34.0	382.9 / 34.4	390.4 / 34.1	401.0 / 34.91	403.1 / 35.2

(b) Distance error (in mm) and 95% confidence (in mm) of a tree with a depth of 8

col \exp	$p > 0.4$	$p > 0.45$	$p > 0.5$	$p > 0.55$	$p > 0.6$	$p > 0.65$	$p > 0.7$	$p > 0.75$	$p > 0.8$	$p > 0.85$
$p < 0.1$	198.6 / 16.6	197.2 / 15.7	199.8 / 16.2	206.9 / 17.5	215.6 / 17.3	229.2 / 19.0	249.6 / 21.5	261.6 / 23.8	294.4 / 29.0	306.0 / 31.2
$p < 0.2$	194.7 / 16.2	194.3 / 15.5	195.5 / 15.5	216.6 / 22.9	227.5 / 21.0	237.0 / 21.7	263.5 / 24.4	276.5 / 26.0	307.1 / 30.6	320.7 / 32.4
$p < 0.3$	196.3 / 16.0	194.7 / 15.1	196.0 / 15.3	218.3 / 22.7	229.6 / 21.0	255.3 / 24.3	275.8 / 26.0	287.4 / 28.0	315.3 / 31.7	325.2 / 32.7
$p < 0.4$	x	214.5 / 20.3	203.5 / 17.3	240.8 / 25.0	261.0 / 27.5	300.7 / 30.5	310.5 / 28.9	325.3 / 31.1	348.8 / 33.4	352.8 / 33.9
$p < 0.5$	x	x	x	276.8 / 30.7	301.8 / 32.9	323.3 / 32.5	329.1 / 31.4	347.0 / 32.5	365.6 / 34.4	369.8 / 34.7
$p < 0.6$	x	x	x	x	x	378.9 / 34.6	379.5 / 35.0	387.3 / 34.7	397.6 / 35.5	400.7 / 35.7

(c) Distance error (in mm) and 95% confidence (in mm) of a tree with a depth of 10

col \exp	$p > 0.4$	$p > 0.45$	$p > 0.5$	$p > 0.55$	$p > 0.6$	$p > 0.65$	$p > 0.7$	$p > 0.75$	$p > 0.8$	$p > 0.85$
$p < 0.1$	197.0 / 16.9	196.4 / 16.0	199.9 / 16.3	199.3 / 17.0	217.9 / 17.6	229.9 / 19.2	249.0 / 21.5	261.4 / 23.8	294.3 / 29.0	306.0 / 31.2
$p < 0.2$	193.6 / 16.5	193.1 / 15.7	197.8 / 16.5	213.0 / 23.0	227.0 / 21.2	237.1 / 21.7	262.6 / 24.4	276.1 / 26.1	306.9 / 30.6	320.2 / 32.4
$p < 0.3$	192.4 / 16.3	193.2 / 15.7	196.6 / 15.6	214.3 / 23.0	228.1 / 21.0	254.8 / 24.4	274.8 / 26.1	287.0 / 28.0	315.2 / 31.7	325.0 / 32.7
$p < 0.4$	x	214.4 / 24.9	202.8 / 17.8	236.0 / 25.2	259.1 / 27.6	299.4 / 30.6	309.6 / 29.0	325.0 / 31.1	348.8 / 33.4	352.8 / 33.9
$p < 0.5$	x	x	x	274.9 / 30.9	300.6 / 33.0	322.7 / 32.6	329.0 / 31.7	347.0 / 32.5	365.5 / 34.4	369.8 / 34.7
$p < 0.6$	x	x	x	x	x	378.8 / 34.6	379.3 / 35.0	387.2 / 34.7	397.6 / 35.5	400.7 / 35.7

(d) Distance error (in mm) and 95% confidence (in mm) of a tree with a depth of 12

col \exp	$p > 0.4$	$p > 0.45$	$p > 0.5$	$p > 0.55$	$p > 0.6$	$p > 0.65$	$p > 0.7$	$p > 0.75$	$p > 0.8$	$p > 0.85$
$p < 0.1$	200.8 / 16.6	199.6 / 15.8	203.2 / 16.3	202.9 / 17.3	221.4 / 18.0	230.4 / 19.3	249.2 / 21.5	261.4 / 23.8	294.3 / 29.0	305.9 / 31.2
$p < 0.2$	197.2 / 16.4	193.7 / 15.6	198.6 / 16.8	213.0 / 23.0	227.2 / 21.3	237.2 / 21.7	262.8 / 24.4	276.1 / 26.1	307.0 / 30.6	320.2 / 32.4
$p < 0.3$	193.7 / 16.3	198.2 / 15.6	197.0 / 15.9	212.9 / 22.5	227.9 / 21.1	254.9 / 24.4	274.9 / 26.1	287.0 / 28.0	315.2 / 31.7	325.0 / 32.7
$p < 0.4$	x	215.6 / 24.9	202.9 / 18.0	235.2 / 25.3	258.9 / 27.7	299.5 / 30.6	309.6 / 29.0	325.0 / 31.1	348.8 / 33.4	352.8 / 33.9
$p < 0.5$	x	x	x	273.7 / 30.9	300.5 / 33.0	322.7 / 32.6	329.0 / 31.4	347.0 / 32.5	365.5 / 34.4	369.8 / 34.7
$p < 0.6$	x	x	x	x	x	378.8 / 34.6	379.3 / 35.0	387.2 / 34.7	397.6 / 35.5	400.7 / 35.7