

Towards Task Oriented Localization

Patric Jensfelt

David Austin

Henrik I. Christensen

Center for Autonomous Systems
Royal Institute of Technology
SE-100 44 Stockholm
{patric,daustin,hic}@nada.kth.se

Abstract. In the course of building a fully autonomous robot platform it is important to look at the computational resources spent by the individual modules. Each of them cannot be greedy, or the overall demand for computational power will be beyond what can be handled on-board. Maintaining an estimate of the pose of a mobile robot is a typical example where we might not always need to run the algorithm at the highest possible rate. This paper deals with the problem of determining how much effort is needed in order to accomplish the localization part of a task. The approach we have taken to the problem is to optimize a cost function that accounts for the cost of sensing and the growth of the uncertainty.

1 Introduction

Many papers have been published on the topic of mobile robot localization as this is one of the basic building blocks of any mobile robot system. Two major paradigms can here be identified, grid based methods originating from the work by Moravec and Elfes [1] and feature based methods, where the environment is modeled as a set of features. Here we will use a *feature based representation* of the environment.

In this paper we will focus on the tracking problem. The tracking problem is the problem of maintaining an estimate of the robot's pose, given a prior estimate of the robots position. For this problem there are almost as many methods as there are robots, motivated by the fact that without being able to keep track of the position no tasks can be solved that requires the robot to move. The simplest, and also the most frequently used method is dead-reckoning. Among the feature based methods we find for example [2, 3, 4, 5, 6].

We will use another angle of attack than normally seen. Typically the algorithms that perform the maintenance of the pose estimate of the robot will be run: i) upon receipt of new sensor data, ii) to consume any available CPU cycles, or iii) whenever the robot has moved some distance to create independence between measurements. Instead we propose to let the update frequency, and along with that the quality of the pose estimate, depend on the task at hand. We chose to call this *task oriented localization*. In a fully autonomous robot, all computations will have to be done on-board. Hence, there will be limited computational resources and thus the algorithms should not use more CPU power than needed. This is especially true when the robot is moving. Before setting out on a mission it would therefore be good to know how much of the resources that have to be spent on keeping track of the position, as this for example might effect the speed of travel that can be used.

The rest of the paper is organized as follows. The next section will give examples to further motivate the approach and highlight some of the difficulties involved. In Section 3 the problem is defined in more detailed and the concept of a region of attraction for a task is introduced. Section 4 give a dynamic programming formulation to the problem. The implementation of the most important parts of the method is described in Section 5. Experimental results are given in Section 6 to illustrate the benefits of the method. The paper is concluded with a summary and suggestions for future work.

2 Motivating Examples

2.1 Example 1: Traversing a Corridor

To illustrate the idea behind task oriented localization we give an example where the robot is to go from one room to another through a long corridor. The corridor is shown in Figure 1, and the task is

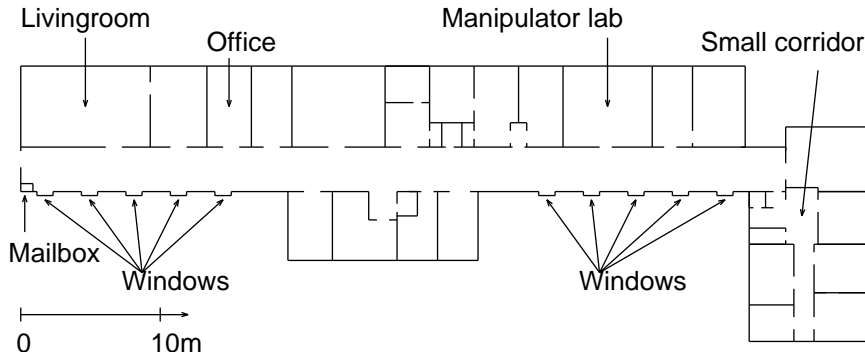


Figure 1: The first floor of the CVAP building.

to go from the Livingroom to the Manipulator lab. Attempting to traverse this distance relying solely on dead-reckoning is doomed to fail as the drift over such a distance is too large. On the other hand if the task is to go to the Manipulator lab, there is no need for the robot to know its position with millimeter accuracy while traversing the corridor. The map shows that the door on the left side of the door to the Manipulator lab is almost 6 meters away and the one on the right is 4 meters away. Hence, if we have a way to detect a door and determine its position reliably, no great accuracy in the localization is needed to be able to find the correct door and pass through it.

2.2 Example 2: Handling Featureless Areas

Another interesting, motivational, example is the case where the robot has to pass an open area or an area which contains no mapped features. When passing through this area we cannot be sure that there are any means to reduce the uncertainty until reaching the other side of the void (see Figure 2). In this case we need to reduce the uncertainty before entering the void sufficiently to be able to pass



Figure 2: Passing through an unknown area.

through it and be able to correctly identify features on the other side. This example is more important than it might seem at a first glance, because a special case occurs where a region only provides one dimensional information. A simple example of this is a corridor with straight, completely bare, walls. Such a corridor provides information in the perpendicular direction but no localization information along the corridor. The uncertainty along the corridor might thus become very large.

2.3 Some of the Difficulties

It is clear that this is a very tough problem. Solving it without constraints would require far more computational resources than running the localization algorithm at highest possible rate. To make an optimal decision, we would have to consider what trajectory to follow and where to face the sensors, how often to update and with what features. Defining what uncertainty can be accepted to accomplish a task, for example going through a door, is also very difficult just based on the environmental model.

3 Problem Definition

The basic question that we face in this paper is to determine how much effort need to be spent on localization to accomplish a particular task, formulated as reaching a certain position. Depending on the task, this might be accomplished based only odometry in some cases. Most often though, external sensors are required.

Guided by the examples above we introduce the so called *region of attraction* for accomplishing a task. The region of attraction is the region inside which we can reliably complete the task. The requirement for the localization system is to reach the goal point with a certain accuracy. On a higher level, a large task can be broken down into smaller subtasks, each with its own region of attraction. The example in Section 2.1, i.e. going from one room to another, might be one subtask when fetching something from a table in the Manipulator lab.

In this paper we will assume that we are given a trajectory to follow, constraints on the size of the uncertainty along the trajectory and what the region of attraction is. The problem is thus to determine which of the visible features along the trajectory to use, when to use them and with what frequency to do the position update. The more often external sensors are used to reduce the uncertainty, the lower the uncertainty becomes, but on the other hand the more computations are needed. The goal of this paper is to come up with a method for determining a good sensing strategy along the trajectory give the constraints. The update rate, ν , is further more assumed to be piece-wise constant. The trajectory from the start position to the end position is split into segments, $k = 0, \dots, M - 1$, with constant update rates, $\nu[k]$.

At our disposal we have a feature based representation of the environment. It is a semi-structured, office type of environment, where all walls are either parallel or orthogonal to each other. The representation is sparse, each room is represented by a low dimensional polygon, typically a rectangle. The map also contains information about where doors are. Additional feature information include the location of sonar based point landmarks. See [7] for more information about the features. We limit ourselves to using the line features.

As a tracking module we use the sample based localization method described in [6], capable of both global localization and pose tracking. Sample based methods have also been used with great success in e.g. [8, 9]. It is well known that the number of samples needed increases with the uncertainty. When no prior knowledge is given, that is every pose is just as likely, many samples are needed. If the pose is known with little uncertainty on the other hand, fewer samples are needed. The computational burden of maintaining the sample set increases with the number of samples, meaning that there are some benefits to being well localized.

4 Dynamic Programming Formulation

The approach we have taken to the problem is to optimize a cost function. The cost function accounts for the cost of sensing and updating the position estimate. The region of attraction is treated as an additional cost. By letting the cost associated with the region of attraction increase with uncertainty, solutions with less final uncertainty becomes beneficial.

The cost function to be minimized can be written in compact form as

$$Q = g(\Lambda[M]) + \sum_{k=0}^{M-1} c(s[k], \mathbf{x}[k], \Lambda[\mathbf{k}]), \quad (1)$$

subject to

$$\Lambda[k + 1] = h(\mathcal{P}, \mathbf{x}[k], \mathbf{x}[k + 1], \Lambda[k], s[k]) \quad (2)$$

$$e(\mathcal{P}, \mathbf{x}[k], \mathbf{x}[k + 1], \Lambda[k]) = 0 \quad (3)$$

where $\{\mathbf{x}[k], \Lambda[k]\}$ denotes the robot pose and pose uncertainty at step $k = 0, \dots, M$. The remaining functions and expressions involved are:

- $s[k]$ is the sensing decision at step k
- $c()$ is the cost of $s[k]$ for a given pose and uncertainty (see Section 5.5).
- $g(\Lambda(M))$ is the cost associated with the region of attraction. A situation where the cost is the same as long as we are inside the region of attraction, but being outside is unacceptable, may be encoded by giving the inside a zero cost and the outside an infinite cost.

- $h()$ is the function that describes how the uncertainty is effected by a sensing decision $s[k]$ when going from $\mathbf{x}[k]$ to $\mathbf{x}[k + 1]$ along a trajectory \mathcal{P} . It depends both of the odometric performance and on the feature characteristics (see Section 5.6).
- \mathcal{P} is the desired trajectory $\mathcal{P} = \mathbf{x}(t)$. The initial pose is $\mathbf{x}(0) = \mathbf{x}[0]$ and the final pose is $\mathbf{x}(T) = \mathbf{x}[M]$.
- $e()$ is a function that describes the constraints on the size of the uncertainty, taking the value 0 for allowed uncertainties.

This problem may be solved using dynamic programming [10] where the state of the system is the uncertainty $\Lambda[k]$. This requires that a discretization of $\Lambda[k]$ be used. Naturally, this is an approximation, but it is required to make the problem tractable. Back-propagation has been used in this paper. In short the idea is that we start from the end and evaluate the function $g(\Lambda[M])$ for all uncertainties. The number of different uncertainties is finite as we have discretized and do not consider infinitely large uncertainties. The result of this is stored. Then we take one step back in time, to $k = M - 1$, and determine what the best sensing decision at that point is for all uncertainties. The function $h()$ tells us what the uncertainty will be at time $k = M$, given a certain sensing decision. Thus the best decision takes into account both the cost for the sensing at time $k = M - 1$ and the cost associated with $\Lambda[M]$. Now we know what decision to make for any given uncertainty at time $k = M - 1$. By taking one step backwards at a time and using the result from the previous step the best sensing strategy can be found given any initial uncertainty.

It is clear that the function $h()$ will be very important in determining what sensing strategy to use. If the features being used are detected with very high accuracy and there are very few false positives, the uncertainty can be reduced significantly with little effort, whereas in a more realistic setting the features are uncertain in position and the number of false positives cannot be neglected.

It is important to remember that there is no such thing as an optimal solution. What we are looking for is the best solution, given a particular discretization, both of the uncertainty and of the trajectory, and on the choice of cost functions. For this reason we will only consider well like regions of attractions, that is, where the cost is the same inside the region, typically zero and is infinite outside. This reduces the number of cost functions to define to one, the function $c()$.

5 Implementation

In this section, the most important parts of the implementation of the method will be described in some detail.

5.1 Odometric Model

For an efficient implementation of this method it is important to have a consistent model of the odometry. By consistent we mean that the resulting uncertainty in the pose of the robot should be independent of in how many steps the update is done and the length of each step. A closed form solution provides a very effective way of calculating the uncertainty.

We assume that the path of the robot can be described as a set of arc motions. Let $\mathbf{u}_k = (D_k, \Delta\gamma_k)$ be the input to the odometric model, where D_k is the distance traveled along an arc and $\Delta\gamma_k$ is the change in motion direction. With this notion the radius of the motion is given by $r_k = \frac{D_k}{\Delta\gamma_k}$. The odometric model can now be written

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) = \begin{cases} \begin{pmatrix} x_k + r_k (\sin(\gamma_k + \Delta\gamma_k) - \sin \gamma_k) \\ y_k - r_k (\cos(\gamma_k + \Delta\gamma_k) - \cos \gamma_k) \\ \gamma_k + \Delta\gamma_k \end{pmatrix} & \Delta\gamma_k \neq 0 \\ \begin{pmatrix} x_k + D_k \cos \gamma_k \\ y_k + D_k \sin \gamma_k \\ \gamma_k \end{pmatrix} & \Delta\gamma_k = 0 \end{cases} \quad (4)$$

We assume that the errors of the two components of the odometric input, \mathbf{u}_k , are uncorrelated and given by $\sigma_{D_k}^2 = k_D D_k$ and $\sigma_{\Delta\gamma_k}^2 = k_\gamma^D D_k + k_\gamma^\gamma$ respectively. k_D , k_γ^D and k_γ^γ are experimentally determined. Without any input from features, the uncertainty thus grows as (first order approximation)

$$\Lambda_{k+1} = \left(\frac{\partial f}{\partial \mathbf{x}_k} \right) \Lambda_k \left(\frac{\partial f}{\partial \mathbf{x}_k} \right)^T + Q_k \quad (5)$$

where Q_k is uncertainty induced by the motion. The elements of the Q_k matrix in (5) are then given by

$$\begin{aligned}
Q_{k,11} &= \frac{1}{2}(k_D|r_k| + r_k^2 k_\gamma) \text{sign}(\Delta\gamma_k) \left(\Delta\gamma_k + \frac{\sin(2\gamma_{k+1}) - \sin(2\gamma_k)}{2} \right) \\
&\quad + r_k^2 k_\gamma \text{sign}(\Delta\gamma_k) (\Delta\gamma_k \cos^2 \gamma_{k+1} - 2 \cos \gamma_{k+1} (\sin \gamma_{k+1} - \sin \gamma_k)) \\
Q_{k,12} &= -\frac{|r_k|k_D + r_k^2 k_\gamma}{4} \text{sign}(\Delta\gamma_k) (\cos(2\gamma_{k+1}) - \cos(2\gamma_k)) \\
&\quad + r_k^2 k_\gamma \text{sign}(\Delta\gamma_k) \left(\frac{1}{2} \Delta\gamma_k \sin(2\gamma_{k+1}) + \cos(2\gamma_{k+1}) - \cos(\gamma_k + \gamma_{k+1}) \right) \\
Q_{k,13} &= |r_k|k_\gamma (\Delta\gamma_k \cos \gamma_{k+1} - (\sin \gamma_{k+1} - \sin \gamma_k)) \\
Q_{k,22} &= \frac{1}{2}(k_D|r_k| + r_k^2 k_\gamma) \text{sign}(\Delta\gamma_k) \left(\Delta\gamma_k - \frac{\sin(2\gamma_{k+1}) - \sin(2\gamma_k)}{2} \right) \\
&\quad + r_k^2 k_\gamma \text{sign}(\Delta\gamma_k) (\Delta\gamma_k \sin^2 \gamma_{k+1} + 2 \sin \gamma_{k+1} (\cos \gamma_{k+1} - \cos \gamma_k)) \\
Q_{k,23} &= |r_k|k_\gamma (\Delta\gamma_k \sin \gamma_{k+1} + (\cos \gamma_{k+1} - \cos \gamma_k)) \\
Q_{k,33} &= k_\gamma |\Delta\gamma_k|
\end{aligned}$$

for an arc motion. A straight line motion, i.e. $\Delta\gamma_k = 0$ give the following Q_k matrix

$$\begin{aligned}
Q_{k,11} &= |D_k| \left(k_D \cos^2 \gamma_k + \frac{1}{3} k_\gamma^D (D_k)^2 \sin^2 \gamma_k \right) & Q_{k,12} &= \frac{1}{2} |D_k| \left(k_D - \frac{1}{3} k_\gamma^D (D_k)^2 \right) \sin(2\gamma_k) \\
Q_{k,22} &= |D_k| \left(k_D \sin^2 \gamma_k + \frac{1}{3} k_\gamma^D (D_k)^2 \cos^2 \gamma_k \right) & Q_{k,23} &= \frac{1}{2} k_\gamma^D |D_k| D_k \cos \gamma_k \\
Q_{k,33} &= k_\gamma^D |D_k| & Q_{k,13} &= -\frac{1}{2} k_\gamma^D |D_k| D_k \sin \gamma_k
\end{aligned}$$

5.2 Splitting the Trajectory

We assumed as stated earlier that we were given a trajectory taking the robot from $x(0)$ to $x(T)$. For ease of computations we have split the path in segment all having the same length. This is not required by the method, but makes the implementation more efficient. The length of the segments should be chosen such so the same features can be seen along a segment.

5.3 Sensing Decisions

As was mentioned in Section 3 the sensing decisions are piece-wise constant, i.e. we make a decision about what features to look for and with what frequency for each segment. To make the problem more tractable we will consider a discrete number of allowed update rates, ν_i . We assume the robot to be moving at a speed of 100 mm/s. The highest update rate at this speed is 1 Hz, constrained in part by computational limitations of the sample based algorithm, but more importantly by the requirement that the robot must move 100 mm, before using a detected feature again. The latter constraint is the standard way of getting higher degrees of independence between the measurements. We have used $\nu_i \in \{0, 0.05, 0.1, 0.2, 0.5, 1\}$ Hz. Different translation speeds can easily be accounted for by a proper scaling of the update rates.

Regarding different features, we distinguish between lines parallel to the x axis and thus providing information about the y coordinate of the robot and those parallel to the y giving the x coordinate of the robot. The different allowed decisions here are $\{0, x, y, xy\}$.

5.4 Discretizing the Uncertainty

The choice of discretization levels will maybe more than anything else effect the outcome of the method, both computationally and in the ability to describe the true performance of the localization. As we have to loop through all possible uncertainties and evaluate all possible sensing decisions, it is of vital importance to keep the number of levels as small as possible.

To reduce the computational burden we discretize the uncertainty independent in x , y and γ . However, when determining the discretization levels the coupling between the three are considered. To be more concrete, the first discretization level is taken to be 0 and the second level is the one

corresponding to the initial uncertainty. The rest of the levels are given by the uncertainty level at the end of each segment when the trajectory is traversed without using any external sensing. If the initial uncertainty is large there will be a large gap between the two first levels. In this case a few extra levels are inserted. It is important to keep in mind that the levels were created using coupling between x , y and γ , but that we henceforth consider the three directions independently. One more measure that can be taken to reduce computations is to remove all uncertainty levels that are larger than the largest allowed along the trajectory.

At this point we might already have the solution to the problem if the final uncertainty with no sensing turns out to be less than what is required to accomplish the task. As we here only consider infinite wells as the final uncertainty cost function, we know that if we end up with a finite cost without doing any sensing, this is the best strategy, because no cost is spent on sensing. No further optimization is needed.

5.5 The c -function

As was stated in Section 3 the number of samples needed increases with the uncertainty. We know from experiments that 10000 samples is sufficient for doing global localization and that 500 samples is enough if the uncertainty is small (tracking). The computational burden involved in doing the pose update is proportional to the number of samples, as well as the update rate and the number features being used. Therefore we suggested the following cost function

$$c(s[k], \mathbf{x}[k], \Lambda[k]) = N(\Lambda[k]) \cdot \nu \cdot \#\text{features}, \quad (6)$$

where N is the number of samples need when the uncertainty is $\Lambda[k]$.

5.6 The h -function

The $h()$ function is implemented as a lookup table, defining the mapping between one discretization level to another. For each uncertainty level the table tells what the resulting uncertainty will be for all possible sensing decisions. This is where the assumption of segments with the same length pays off, as we can use the same lookup table for all segments along the trajectory.

The basis for the lookup table are two functions that describe how the uncertainty in x or y changes with the sensing rate if the corresponding line is used and ditto for the γ uncertainty. These functions depend on the feature characteristics. We have used the same model as in [7] for the lines. The functions are parameterized by initial uncertainty σ_{init}^2 , update rate ν , segment length D , and the uncertainty resulting when no sensing is done σ_{nosens}^2 . The latter is given by the next, higher, discretization level by construction.

$$\begin{aligned} \sigma_{x/y,res}^2 &= \Psi_{x/y} \left(\sigma_{x/y,init}^2, \nu, \sigma_{x/y,nosens}^2 \right) \\ \sigma_{\gamma,res}^2 &= \Psi_{\gamma} \left(\sigma_{\gamma,init}^2, \nu, \sigma_{\gamma,nosens}^2, D \right) \end{aligned}$$

Examples of predictions made by these function can be seen in Figure 3. The functions are continuous and not dependent on the discretization levels. Before building the lookup table h , three small lookup tables are created. The Ψ function for x , y and γ are evaluated for all uncertainty levels and sensing decisions and the resulting uncertainty is taken to be the closest uncertainty level larger than or equal to what is given by the Ψ function. The lookup table h is then constructed by combining the three small lookup tables into one.

6 Experimental Results

To illustrate the ability of the method we give an example that is based on the examples in Sections 2.1 and 2.2. The robot is given the task to go from one side to the other of a long corridor. The sensor is assumed to be facing the left side of the corridor at the start position, rotating with constant angular velocity to face the right side wall at the end position. We must arrive at the end position with less than 200 mm in standard deviation in both x and y . All solutions that fulfill this are equally good and thus the function $g()$ is zero inside this region and infinite outside. To further complicate matters there is a region in the middle of the corridor without any known features. To assure that we arrive at the other side of the corridor we demand that the standard deviation be less than the width of the corridor when entering it. Along the rest of the trajectory the uncertainty is never allowed to

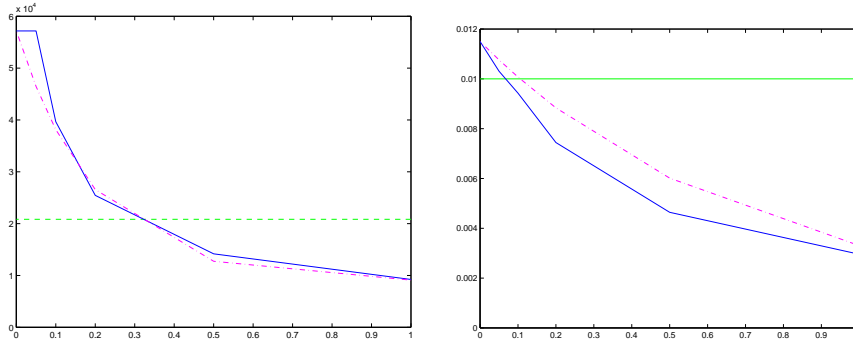


Figure 3: The figures show an example of how the predicted change in uncertainty by the Ψ functions (dash-dotted lines) compares to the simulated result (solid lines). The horizontal lines corresponds to the initial uncertainties. The x-axis is update rate in Hz and the y-axis is variance in mm^2 and rad^2 respectively.

grow larger than that the known obstacles (the walls) stay outside the area defined by one standard deviation. The walls are assumed to be visible if they cover more than 30° field of view and the sensor is inside the ends of the line. The latter constraint is motivated by the fact that open doors and other obstacles are otherwise very likely to occlude the view. Figure 4 show the best sensing strategy given

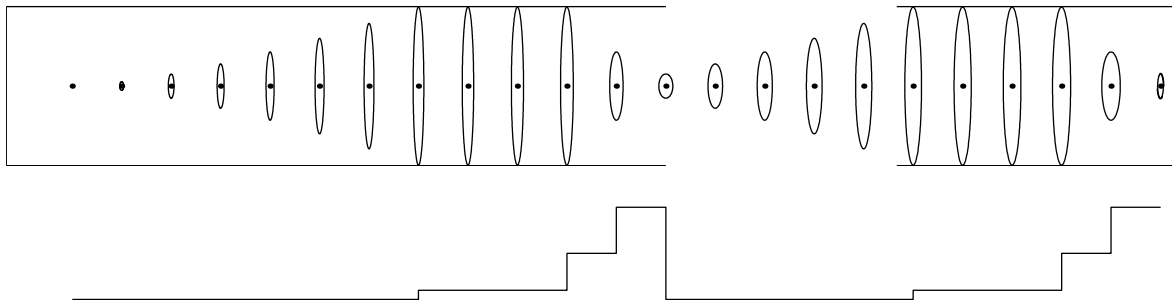


Figure 4: The upper figure shows the uncertainty ellipse at the end of each segment along the path. The figure is stretched vertically for illustrational purposes. The lower figure shows the sensing rate that is used along the segments. The maximum sensing rate corresponds to 1 Hz update.

that the path is divided into 3 m long segments. The resulting strategy is very intuitive.

1. Do not do any sensing until we have to and then just enough to stay inside the area defined by the corridor walls.
2. Before entering the void we step up the sensing to leave the known area with small enough uncertainty to be able to enter the other side of the corridor.
3. On the other side only minimum sensing is done until we come close to the target point when the sensing is increased again.

That this is a sound solution is clear. Sensing is only done enough to compensate for the odometric drift and the uncertainty is only decreased close to where it has to be small so as not to work against the odometric drift unnecessarily much. Had the void been made larger, the method would be able to give the answer that it cannot be passed reliably. The same is true if there would have been no wall at the right side of the corridor, as the uncertainty could not have been decreased enough to enter the region of attraction.

7 Summary and Future Work

In this paper we have presented task oriented localization as a framework for minimizing the resources spent on localization. This is an important problem as a fully autonomous system requires on-board computing, and therefore has limited resources. In this paper, a set of assumptions have been made that make the problem tractable, without reducing it to a “toy” problem. We have chosen a dynamic programming approach for solving the optimization problem.

Clearly there are many open problems that need to be solved, for example how do we divide large tasks into smaller subtasks. The function $e()$ should be chosen in a way as to reduce the risk of generating multiple hypotheses caused by ambiguities resulting from a large uncertainty. The environmental model must be the basis for this design.

8 Acknowledgment

This research has been sponsored by the Swedish Foundation for Strategic Research through the Center for Autonomous Systems. The funding is gratefully acknowledged.

References

- [1] H. Moravec and A. Elfes, “High resolution maps from wide angle sonar,” in *Proc. of the International Conference on Robotics and Automation*, pp. 116–121, IEEE, 1985.
- [2] J. Leonard and H. Durrant-Whyte, “Mobile robot localization by tracking geometric beacons,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.
- [3] J. Crowley, “Navigation for an intelligent mobile robot,” *IEEE Journal of Robotics and Automation*, vol. 1, pp. 31–41, Mar. 1985.
- [4] P. Jensfelt and H. Christensen, “Laser based pose tracking,” in *Proc. of the International Conference on Robotics and Automation*, (Detroit, Michigan, USA), IEEE, May 1999.
- [5] O. Wijk and H. Christensen, “Extraction of natural landmarks and localization using sonars,” in *Proc. of the International Symposium on Intelligent Robotic Systems*, (Edinburgh, Scotland), July 1998.
- [6] P. Jensfelt, D. Austin, O. Wijk, and M. Andersson, “Feature based condensation for mobile robot localization,” in *Proc. of the International Conference on Robotics and Automation*, (San Francisco, CA, USA), May 2000.
- [7] P. Jensfelt, O. Wijk, D. Austin, and M. Andersson, “Experiments on augmenting condensation for mobile robot localization,” in *Proc. of the International Conference on Robotics and Automation*, (San Francisco, CA, USA), May 2000.
- [8] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, “Monte carlo localization - efficient position estimation for mobile robots,” in *Proc. of the National Conference on Artificial Intelligence*, 1999.
- [9] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, “Monte carlo localization for mobile robots,” in *Proc. of the International Conference on Robotics and Automation*, vol. 2, pp. 1322 – 1328, May 1999.
- [10] D. P. Bertekas, *Dynamic Programming and Optimal Control, vol 1*. Athena Scientific, 1995.