2012 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 7-12, 2012. Vilamoura, Algarve, Portugal

Active Robot Localization with Macro Actions

Koosha Khalvati and Alan K. Mackworth

Abstract— **We propose a novel method for solving the localization problem, particularly useful in self-similar environments. Localization is the task of determining a robot's position with little or no prior knowledge about its initial location. We introduce a computationally efficient active strategy for the localization task. The proposed method also automatically generates optimal macro actions giving the robot the ability to localize in all environments. Combining the active strategy and macro actions, the method introduces an efficient strategy for determining a robot's position in any environment where localization is possible.**

I. INTRODUCTION

Localization is an essential task in robotics. A robot cannot be called autonomous without the ability to localize itself. Furthermore, a robot is often unable to perform other tasks without being aware of its location. A robot's ability to localize itself accurately and efficiently is the subject of numerous research papers in recent years. It has been proven that localizing with minimum cost is an NP-hard problem [2]. Therefore, researchers have sought to find near optimal solutions efficiently. Active strategies are good candidates to fulfill these two criteria: near optimality and efficiency. Some papers covering these strategies have defined heuristics such as taking the robot to places with the maximum number of new features [9] or choosing the actions that lead to minimum expected number of possible states [1] Another active approach is entropy-based that chooses the action that minimizes the expected entropy [3]. In this paper, we introduce a new active strategy for the localization problem, that can be efficiently computed.

Our main contribution, however, is providing an algorithm to deal with the problem of self similarity of environments. This problem is that, due to the self similarity of the environment, the robot gains no further information by performing basic actions. Examples of these situations are corridors, identical offices and big rooms with no objects in them when the robot uses a limited range laser sensor. In the first two cases, the robot merely sees similar walls and in the third, it finds nothing in its laser range. If the robot uses vision instead of a laser, it may find many similar places in environments such as hospitals or schools. Many rooms look the same in hospitals. Also, classrooms and offices may have the same pattern in schools and the robot cannot localize itself until it exits a room. In these cases, the robot should perform a sequence of actions instead of just one. In this paper, we propose an algorithm to generate optimal cost sequences that can reduce the ambiguity in any situation. Also, this algorithm can be used to determine whether or not a robot can localize itself in an environment.

This work was supported by NSERC, Canada Research Chairs and ICICS.

K. Khalvati and A.K. Mackworth are both with the Department of Computer Science, University of British Columbia, Vancouver, B.C. V6T 1Z4 Canada {kooshakh,mack}@cs.ubc.ca

II. PREVIOUS WORK

Proof of NP-hardness of finding the optimal strategy for localization problem was provided by Dudek *et al.* in 1995 [2]. After that, Tovey and Koenig [15] proved that even minimizing the cost with factor $c \log n$ is NP-hard. So researchers tried to find near optimal solutions. Koenig *et al.* [10] proposed an $O(\log^3 n)$-factor algorithm on grid-based maps and extended it to polygonal maps. The problem of this algorithm is that its high computational cost ($\Omega(n^{12})$) allows it to be used only in very small environments [16]. Rao *et al.* [14] also suggested a near optimal randomized algorithm in polygonal maps. Both of these algorithms assumed that actions and sensors are noise-free and hence their application in real environments is still an open issue.

On the other hand, methods proposed for real environments address concerns with noise and computational efficiency. Although researchers in this area tried to minimize the action cost by choosing useful heuristics, they did not discuss the bounds of the cost of the solutions that are found by their active strategies. Jensfelt *et al.* [9] proposed a strategy based on heuristics of going to the places with a maximum number of new features and avoiding getting to previous positions. In the method proposed by Gasparri *et al.* [5], the robot moves to the nearest obstacle in each step. Fox *et al.* [3] proposed an active strategy that chooses the action that minimizes the expected entropy in each step. Porta *et al.* [13] improved the computational cost of their method by using particle filters. Murtra *et al.* [1] proposed another method that minimizes the expected number of hypotheses instead of entropy at each step. The computational cost of their algorithm was less than the two previous algorithms that used entropy.

To solve the problem of self similarity, Fox *et al.* [3] put some relative target points among their actions. The path planning from the current states to the target points is done by the robot, but the target points themselves are given by the domain expert. In the method proposed by Murta *et al.* [1], some random target points were generated in a specific range from possible locations and the selection process was performed among those points. The range should be large enough to give the robot the ability to localize in all circumstances and, in the larger ranges, many target points are needed. This would increase the computational cost. The method we propose generates all macro actions automatically and specifies the whole path instead of just target points. We

call these paths macro actions. Macro action generation is the subject of some work in related fields as well. Work by He *et al.* [8][7] and Kurniawati *et al.* [11] in planning under uncertainty and the work of McGovern [12] in reinforcement learning are examples of macro action generation. However, their approaches for generating macro actions and the reason for using them are quite different from ours.

III. Markov Localization

In a localization problem, it is assumed that the map is given to the robot. The map is represented as a finite set of states, S. The state of the robot at step k is x_k. Following [4], we assume that the system is Markovian, in that the new state of the robot, only depends on the previous state and the action it took in that state. In each state, the robot has an observation in the set of all possible observations, O. The observation at step k is z_k and the sequence of all observations up to step k is Z_k. A is the set of all possible actions which the robot can perform. Each action $a \in A$ in state s has a positive cost, $C(s, a)$. In addition, the action performed by the robot at step k is u_k and the sequence of all actions performed up to step k is U_k. During localization, the robot's belief about being in state s is $Bel(x_k = s)$.

$$Bel(x_k = s) = p(x_k = s|U_{k-1}, Z_k) \tag{1}$$

As the system is Markovian, this probability is:

$$Bel(x_k = s) = p(x_k = s|u_{k-1}, x_{k-1}, z_k) \tag{2}$$

Localization consists of three steps: (i) Action selection, (ii) Performing the action and updating the belief based on that action, and (iii) Observing and updating the belief based on that observation. At first, we put aside step (i) and explain the other two steps assuming that the action has been selected. Suppose the robot is in x_{k-1} when it performs action u_{k-1}, the belief states will be updated as below. Belief is shown by B^- in this step because only u_{k-1} and x_{k-1} are considered.

$$Bel^-(x_k = s) = \sum_{x_{k-1}} p(x_k|x_{k-1}, u_{k-1})Bel(x_{k-1} = s) \tag{3}$$

Since $Bel^-(x_k)$ depends on $Bel(x_{k-1})$, $Bel(x_1)$ should be given. If the robot has some prior knowledge about its location, $Bel(x_1)$ is defined. On the other hand, if the robot knows nothing, the probability density of $Bel(x_1)$ is uniform across all states. That is global localization.

For step (iii), when the robot observes z_k, the belief states will be updated, using Bayes theorem:

$$Bel(x_k = s) = p(z_k|x_k = s) \times Bel^-(x_k = s)/p(z_k) \tag{4}$$

These updates belong to step (ii) and step (iii). Even with a random strategy in the action selection step, the robot may localize itself after some actions. But, that accomplishment is not guaranteed. Even if the robot succeeds in the localization task, it may take many actions with far higher cost than the optimal strategy. Finding the optimal strategy is NP-hard [2], so we attempt to find a near-optimal active strategy to reduce the cost as much as possible. In the next section we explain our active strategy for action selection.

IV. An Active strategy for action selection

A. Additional Variables

Considering the previous variables which are necessary for the Markov localization problem, we need to define some additional variables for our strategy based on those. The new variables are: the observation difference between two states, the transition function between states assuming the actions are deterministic, and the cost of an action when we deal with two states instead of one.

$d(s_i, s_j)$ is the *observation difference* between two states s_i and s_j calculated as:

$$d(s_i, s_j) = \frac{1}{2} \sum_{o \in O} [p(o|s_i)(1-p(o|s_j)) + p(o|s_j)(1-p(o|s_i))] \tag{5}$$

Two states s_i and s_j are $distinguishable$ if and only if $d(s_i, s_j)$ is greater than a threshold γ. We should set a suitable value for γ based on the noise in the observations. This value is usually very close to the probability of the most probable observation in a state.

Observation difference is a number in $[0, 1]$. When observation o has a high probability in s_i and a low probability in s_j, $p(o|s_i).(1-p(o|s_j))$ would be close to 1 and when o has high probability in both states, $(1-p(o|s_j))$ is very small and as a result $p(o|s_i).(1-p(o|s_j))$ is close to 0. So for the pairs with similar high probability observations $d(s_i, s_j)$ is close to 0 but for the states with different observations, it would be close to 1. Thus, when $d(s_i, s_j)$ is above a threshold, for most of the expected observations s_i and s_j can be distinguished from each other with high probability.

Let $f^*(s, a)$ be the state that the robot goes to from state s with action a in the noiseless case. We can assume that this state is the most probable state that the robot goes to:

$$f^*(s, a) = argmax_{s'} p(s'|s, a) \tag{6}$$

If there is more than one s' with maximum probability, we choose one of them arbitrarily.

Last, we define cost of an action for two states as:

$$C(s_i, s_j, a) = max(C(s_i, a), C(s_j, a)) \tag{7}$$

B. Explanation of Algorithm for Action Selection

The original idea for the strategy we are using, comes from work in the field of active learning by Golvin *et al.* [6]. They called their algorithm EC^2 (Equivalence Class Edge Cutting). To explain our strategy, we should define a graph. The vertices of this graph are the states and the edges are the actions. Two states s_i and s_j are connected to each other by action a if and only if $f^*(s_i, a)$ and $f^*(s_j, a)$ are distinguishable. The weight of the edge is:

$$w(s_i, s_j, a) = Bel(s_i) \times Bel(s_j) \times \\ min(p(f^*(s_i, a)|s_i, a), p(f^*(s_j, a)|s_j, a))/C(s_i, s_j, a) \quad (8)$$

If the robot assigns zero belief to either of the two states, the weight of the edge would be 0, so there is no need to calculate the weight. We only consider the edges between states with positive belief functions.

For each action a, $w(a)$ is:

$$w(a) = \sum_{s_i, s_j} w(s_i, s_j, a) \quad (9)$$

In each step of action selection, we choose the action with the highest assigned weight:

$$a^* = argmax_a w(a) \quad (10)$$

The algorithm is shown below as Algorithm 1.

C. Intuition

The intuitions behind the concept of weight and the action selection criteria are explained most easily by an example. Each edge in the graph means the action assigned to the edge can remove at least one of the two vertices from belief states (not the states themselves but their future states, f^*). So, the action with highest weight has most power to distinguish between states, especially the ones with higher probability.

Algorithm 1: The algorithm for action selection with basic actions

input : Belief states $Bel(.)$, actions A and their costs C, observation differences of all pairs d, and transition function f^*

output: a^*, the action to be performed by the robot

```
foreach a ∈ A do w(a) = 0
foreach s_i, s_j, a do
    if d(f*(s_i, a), f*(s_j, a)) > γ then
        w(s_i, s_j, a) = Bel(s_i) × Bel(s_j) ×
        min(p(f*(s_i, a)|s_i, a), p(f*(s_j, a)|s_j, a))
        /C(s_i, s_j, a)
    else w(s_i, s_j, a) = 0
end
foreach s_i, s_j, a do w(a) = w(a) + w(s_i, s_j, a)
Select a* = argmax_a w(a)
```

1	1	0	1	1	0
0	S1	1	1	0	1
1	0	0	S3	1	0
1	S2	1	1	S4	0
0	0	0	1	1	0
0	1	1	0	1	0

Fig. 1. Example 1. The map and the observations in various states

Consider the map in Fig. 1. The robot is initially in one of the shaded squares, $s1$ to $s4$, and the beliefs of the robot are:

$$Bel(s1) = 0.2 \\ Bel(s2) = 0.2 \\ Bel(s3) = 0.1 \\ Bel(s4) = 0.5$$

We have an observation in each square, 0 or 1, as shown in the map. In addition, the observation in s1 to s4 is 0. We have four actions: up, down, right and left, all with unit costs. The actions and observations are noise free. With these assumptions we would like to localize the robot with this active strategy. Which action should be chosen next? To answer this question we build the graph (Fig. 2). For example, if the robot performs 'up', it will observe 1 if it is in $s1$ and it will observe 0 if it is in $s2$. So after performing up at least one of these states would be removed. The weights of the actions are :

$$w(up) = Bel(s1) * Bel(s2) + Bel(s2) * Bel(s3) + \\ Bel(s2) * Bel(s4) = 0.16$$
$$w(down) = Bel(s1) * Bel(s3) + Bel(s1) * Bel(s4) + \\ Bel(s2) * Bel(s3) + Bel(s2) * Bel(s4) = 0.24$$
$$w(right) = Bel(s1) * Bel(s4) + Bel(s2) * Bel(s4) + \\ Bel(s3) * Bel(s4) = 0.25$$
$$w(left) = Bel(s1) * Bel(s2) + Bel(s1) * Bel(s4) + \\ Bel(s2) * Bel(s3) + Bel(s3) * Bel(s4) = 0.21$$

So $right$ would be chosen. After performing this action either the robot would know its exact position by observing 0 or a state with high probability would be removed from the belief states. In the latter case, as the state was not a correct hypothesis, this elimination is a significant achievement. On the other hand, if the robot performs up, with the chance of 0.8 it would only eliminates a state with probability of 0.2. No matter what the robot observes, actions $down$ and $left$ reduce the number of states with positive probability to 2. This reduction is a good improvement, but the high prior probability of $s4$ leads the algorithm to choose $right$.

The presence of $min(p(f^*(s_i,a)|s_i,a), p(f^*(s_j,a)|s_j,a))$ in the formula, is a way to deal with noise. An action with greater noise has less chance of being selected because the states would be removed from belief set only if the action is performed accurately. We also consider the cost of the action in our formula, because our goal is to perform the localization task with the least possible cost.

In the active learning field, Golovin *et al.* proved that the cost of the policy that is generated by EC^2 is near optimal and its bound is:

$$C(\pi_{EC^2}) \leq (2\ln(1/p_{min})+1)C(\pi^*) \tag{11}$$

In their problem, the algorithm tries to select the correct hypothesis among many by choosing a subset of tests with minimum cost from a set of all possible tests. π^* is the optimal policy and p_{min} is the probability of the least probable hypothesis. In that problem tests are performed without noise and the weights only contain probabilities of two hypotheses and their cost [6].

V. Macro Actions

There are many situations, where because of the symmetry in the environment, none of the actions are useful for localization. In these cases, we need a sequence of actions instead of a single basic action. In this part we propose an algorithm that generates minimum cost sequences of actions that solve the problem of localization no matter where the robot is. We call these sequences macro actions.

Our strategy of action selection is based on the actions that distinguish between each of a pair of states. So if there is a situation where no action can be selected, it means that there is no action for any pair of states. Therefore, we build the macro actions based on these pairs. In other words, for any pair of states, we find a sequence of actions that can distinguish between them. In generating these macro actions the actions are assumed to be deterministic, so only f^* is used.

A. The Macro Action Algorithm

Algorithm 2 generates macro actions. For each pair of states, we should find a macro action that can resolve the localization problem. This macro action is called $macro\_action(s_i,s_j)$ and its cost is $C(macro\_action(s_i,s_j))$. First we set $macro\_action$ empty for all pairs. Then, for each pair of s_i and s_j that are distinguishable, $macro\_action(s_i,s_j)$ is set to zero with zero cost, which means that without any action we can distinguish between them. In the next step, we look at each pair with an undefined macro action (Line 11) and for each basic action a we check whether $macro\_action(f^*(s_i,a), f^*(s_j,a))$ is defined or not (Line 12). If it is, we can build a path for s_i and s_j by adding a to the head of $macro\_action(f^*(s_i,a), f^*(s_j,a))$ (Line 24). The cost of the new path would be $max(C(s_i,a), C(s_j,a)) + C(macro\_action(f^*(s_i,a), f^*(s_j,a)))$ (Line 25). However, we do not set this macro action at the first step. We merely calculate the cost for all pairs and only set the macro actions of those with minimum costs (Line 23). After the first step of calculating new macro actions for all pairs and setting the minimum sequences, we repeat this step and recalculate the paths, find minimum ones (Line 14 and line 18) and update them (Lines 23-26). We repeat this step until no macro action can be built. Most of the time this would happen because we have already built the macro actions for all of the pairs, but there may be situations when there exist two states for which we cannot find a sequence. In this situation, we can say that if the robot is in one of these states, it can never localize itself. In fact, our algorithm can be used as a test to see whether a robot can always localize itself in the map.

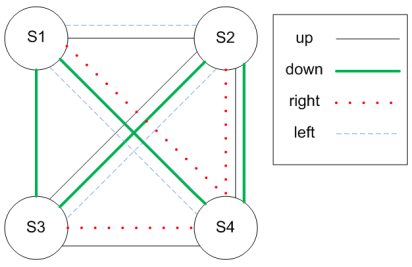

Fig. 2. Example 1. The graph that is built in our strategy

Algorithm 2 generates exactly one action sequence for each pair of states. This sequence is optimal in deterministic environments and near-optimal in noisy environments. The proof of optimality of these sequences in deterministic environments is given in the Appendix.

VI. Performing Active Localization with Macro Actions

The localization algorithm with macro actions, Algorithm 3, is quite similar to Algorithm 1. This time, we consider not only the basic actions of the robot, but also macro actions that were generated previously. To consider the macro actions in the graph we should define f^* and the cost for them. If we have a macro action a' that is a sequence of basic actions $\langle a_1, a_2, ..., a_l\rangle$, then $f^*(s,a')$ is calculated by this recursive formula:

$$f^*(s,\langle a_1,...,a_l\rangle) = f^*(f^*(s,\langle a_1,...a_{l-1}\rangle), a_l) \tag{12}$$

The cost of this macro action for states s_i and s_j is calculated the same way we calculate costs in generating macro actions:

$$C(s_i,s_j,\langle a_1,...a_l\rangle) = max(C(s_i,a_1), C(s_j,a_1)) + C(f^*(s_i,a_1), f^*(s_j,a_1), \langle a_2,...,a_l\rangle) \tag{13}$$

Also, $p(f^*(s,a')|s,a')$ is obtained by the following formula:

$$p(f^*(s,\langle a_1,...,a_l\rangle)|s,\langle a_1,...,a_l\rangle) = p(f^*(s,a_1)|s,a_1)\times p(f^*(f^*(s,a_1),\langle a_2,...,a_l\rangle)|f^*(s,a_1),\langle a_2,...,a_l\rangle) \tag{14}$$

Algorithm 2: The algorithm for generating macro actions

input : Set of states S, actions A and their costs C, observation differences of all pairs d, and transition function f^*
output: Macro actions for all pairs of states

MAX = very large value
foreach s_i, s_j **do** $macro\_action(s_i, s_j) = \emptyset$
foreach s_i, s_j *that* $d(s_i, s_j) > \gamma$ **do**
  $macro\_action(s_i, s_j) = 0$
  $C(macro\_action(s_i, s_j)) = 0$
end
$min\_set = \{(s_1, s_2, a_1)\}$ //set this value just to enter the while loop at first
while $min\_set \neq \emptyset$ **do**
  $min\_set = \emptyset$
  $min\_cost = MAX$
  foreach s_i, s_j, a *that* $macro\_action(s_i, s_j) = \emptyset$ **do**
    if $macro\_action(f^*(s_i, a), f^*(s_j, a)) \neq \emptyset$ **then**
      if $C(s_i, s_j, a) + C(macro\_action(f^*(s_i, a), f^*(s_j, a))) = min\_cost$ **then**
        add (s_i, s_j, a) to min_set
      end
      if $C(s_i, s_j, a) + C(macro\_action(f^*(s_i, a), f^*(s_j, a))) < min\_cost$ **then**
        Set min_set empty
        add (s_i, s_j, a) to min_set
        $min\_cost = C(s_i, s_j, a) + C(macro\_action(f^*(s_i, a), f^*(s_j, a)))$
      end
    end
  end
  foreach (s_i, s_j, a) *in* min_set **do**
    $macro\_action(s_i, s_j) = \langle a, macro\_action(f^*(s_i, a), f^*(s_j, a)) \rangle$
    $C(macro\_action(s_i, s_j)) = min\_cost$
  end
end

The other parts of the algorithm are exactly the same. One important thing to note about performing a macro action is that if a macro action is selected during the action selection part, the robot performs it step by step. This means that after performing each single action of sequence, it observes the environment and updates its beliefs. If the observations are the same as robot's expectation (the action has been performed accurately), it will continue executing the sequence; if not, it ignores the rest of the sequence and the action selection method is called again. The algorithm is shown below as Algorithm 3.

VII. Implementation and Computational Cost

Our method puts most of the computational work into the offline computation that is done only once per map and can be used many times. The most time consuming algorithm in this computation is the algorithm for macro action generation. On the other hand, we try to make the online part of our method as fast as possible and use a common technique to reduce its time complexity.

A. Offline Computation

The first precomputation is calculating $p(o|s)$ for all observations and states and putting them in a lookup table as in [3]. This lookup table reduces the time needed for updating beliefs after an observation and also finding distinguishable pairs. The second precomputation is generating macro actions for all pairs of states (Algorithm 2). This algorithm is quite expensive and in the case where all actions have unit cost, it takes $O(|S|^2 \times k)$ where k is the maximum length optimal path required to distinguish two states. In the worst case k would be $O(|S|)$ and the total time needed is $O(|S|^3)$. But usually it is far less than $|S|$. For actions with different costs, if the cost of the maximum cost action is C_{max} and the cost of minimum cost action is C_{min}, in the worst case, the computational cost would be $O(|S|^3 \times C_{max}/C_{min})$. For each pair the macro action should be stored and this requires $|S|(|S|-1)/2$ memory. For memory usage efficiency, we need only store the macro actions for the states that are not distinguishable, not storing empty sets.

B. Online Computation

In the online stage of the method, if there are n states with positive probability, action selection takes $O(n^2 \times |A'|)$ where A' is the set of candidate actions and macro actions. A common method for reducing computational cost is to assume the probability density is a mixture of Gaussians [3] [1]. To do this, states with a probability higher than a threshold would be selected and are assumed to be the mean

Algorithm 3: The algorithm for action selection with macro actions

input : Belief states $Bel(.)$, actions A and their costs C, observation differences of all pairs d, and transition function f^*, and macro actions
output: a^*, the action or the sequence of actions that should be performed by the robot

$A' = \emptyset$
foreach s_i, s_j *that* $Bel(s_i) \times Bel(s_j) > 0$ **do** add $macro\_action(s_i, s_j)$ to A'
foreach $a' \in A'$ **do** $w(a') = 0$
foreach s_i, s_j, a' **do**
  if $d(f^*(s_i, a;), f^*(s_j, a')) > \gamma$ **then**
    $w(s_i, s_j, a') = Bel(s_i) \times Bel(s_j) \times min(p(f^*(s_i, a')|s_i, a'), p(f^*(s_j, a')|s_j, a')) / C(s_i, s_j, a')$
  else $w(s_i, s_j, a') = 0$
end
foreach s_i, s_j, a' **do** $w(a') = w(a') + w(s_i, s_j, a')$
Select a^* which $a^* = argmax_{a'} w(a')$

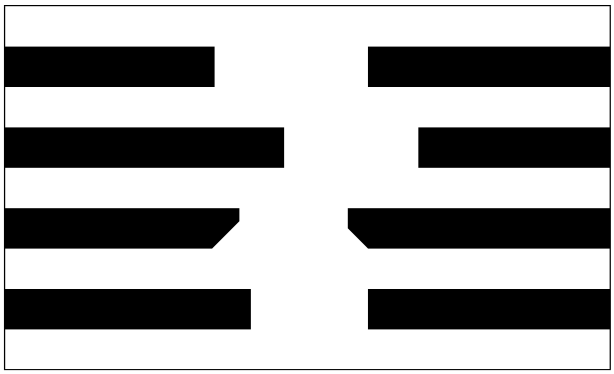

Fig. 3. The map of the self similar environment used in the experiment. The environment is $30m \times 18m$. The width of the corridors is $2m$ and their maximum length is $14m$.

TABLE I

FRACTION ACCOMPLISHED IN THE LOCALIZATION TASK WITH FEWER THAN 500 BASIC ACTIONS WITH AND WITHOUT MACRO ACTIONS

Method	Fraction accomplished
Macro actions	100%
Basic actions and random walk	86.2%

TABLE II

NUMBER OF AVERAGE BASIC ACTIONS FOR LOCALIZATION WITH AND WITHOUT MACRO ACTIONS

Method	Average number of basic actions
Macro actions	25.61
Basic actions and random walk	79.13

of Gaussian densities that model probability of all states. If the number of these means is m, the complexity of the action selection is $O(m^2 \times |A'|)$. This computational cost is $O(n \times m \times |A|)$ for entropy based localization [3] and as $m \ll n$ our approach is much more efficient.

In the case where the robot has no prior knowledge about its position (global localization), to avoid high computational cost the robot selects the first few (3 to 10) actions randomly because the number of possible states is very large, but it would drastically reduce in the first few steps.

VIII. EXPERIMENTS

We simulated a robot with a noisy laser range finder in a $30m \times 18m$ self similar environment, shown in Fig. 3. It has long corridors ($14m$) that are usually the main cause of similarity in real environments for laser range finder robots. The angular resolution of the range finder is 1° and its maximum angular range is 240°. The maximum laser range is $10m$. The basic actions are going $20cm$ forward, backward, left and right and also turning 5° clockwise or counterclockwise, all with unit costs. All actions are performed accurately only 85% of the time. The map is grid-based and the grid size for the state space is $(20cm, 20cm, 5^\circ)$. As a result, there exist 2,881,196 pairs of indistinguishable states.

Our macro action generation method generated 6,073 macro actions with lengths of 1 to 31 for this environment. To show the effect of macro actions, we tested our method with and without macro actions. In the case where we only used basic actions, when there was no action able to reduce uncertainty, a random basic action was generated. In both tests the robot is considered localized, if and only if its certainty about its position is greater than 0.95 or the number of basic actions performed exceeds 500. We ran 1000 trials of global localization and tested both strategies in each trial. In each trial, the initial location was selected randomly from the locations which have the same observation in the noiseless case with at least one other location, meaning that the robot has to perform at least one action to accomplish the localization task. The strategy of using macro actions, passed all the trials in less than 500 actions. The other strategy, failed to accomplish the task in less than 500 actions, in 138 trials (Table I). Moreover, in the trials where both strategies accomplished the task, the macro action strategy performed the localization task significantly better than the other one on average, using fewer than one third the basic actions. The average number of basic actions that are needed for localization in successful trials is shown in Table II.

IX. DISCUSSION

In this paper, we have introduced a new active strategy for localization and macro actions that are generated automatically by the robot, giving it the ability to localize itself in any environment where localization is possible. Our method is based on Markov localization. One of the drawbacks of Markov localization is that we need to represent states explicitly and a large number of states is needed for large environments especially for 3D navigation. As a result our method is more useful for 2D navigation in small to medium-sized self-similar environments where single action based methods do not work.

APPENDIX

A. Proof of Correctness of the algorithm for generating macro actions in deterministic environments

Theorem: In a deterministic environment, for any pair of states, Algorithm 2 finds the minimum length macro action that can distinguish between them.

Proof: We do this by induction:

Basis: The algorithm finds the macro action for the states that are distinguishable by zero cost.

As costs of all actions in any state are positive, zero cost means that the states are distinguishable and there is no need for macro action. We set these macro actions to zero in the line 3 of Algorithm 2 which means there is no need for a macro action. Therefore, costs and macro actions of all of those pairs are defined initially in our algorithm.

The inductive step: Assuming that macro actions of all pairs with minimum cost macro action of less than cost P are defined and no other macro action is set, the macro action of pairs with minimum cost macro action of cost P will be defined in the next step.

If the minimum cost macro action of s_i and s_j costs P, first of all it is not defined yet because only macro

actions with cost of less than P have been defined. If this macro action is a single action a with cost P, it will be defined in this step because $f^*(s_i, a)$ and $f^*(s_j, a)$ are distinguishable and macro actions of all distinguishable pairs are proven to be set in basis. Macro action of (s_i, s_j) will be set to a attached to $macro\_action(f^*(s_i, a), f^*(s_j, a))$ which is zero, meaning no action. And if this macro action is not a single action we show it by $\langle b_1, b_2, ..., b_{k-1}, b_k \rangle$ that each b_i is one basic action. As all actions have positive cost, $C(s_i, s_j, b_1)$ is positive and $(C(f^*(s_i, b_1), f^*(s_j, b_1), \langle b_2, ..., b_k \rangle))$ is less than P. Thus $macro\_action(f^*(s_i, b_1), f^*(s_j, b_1))$ has been already defined and as both $\langle b_2, ..., b_k \rangle$ and $macro\_action(f^*(s_i, b_1), f^*(s_j, b_1))$ are minimal, the costs of them are exactly equal. $macro\_action(f^*(s_i, b_1), f^*(s_j, b_1))$ is minimal by the induction assumption and $\langle b_2, ..., b_k \rangle$ is minimal because if it is not, we can put the minimal path after b_1 and find another path with less cost for distinguishing between s_i and s_j. This contradicts the assumption that the minimum cost macro action of s_i and s_j costs P. So macro action of s_i and s_j will be defined by attaching b_1 to $macro\_action(f^*(s_i, b_1), f^*(s_j, b_1))$ with cost P.

As a macro action with cost P is defined, all macro actions with cost of greater than P are not minimum and would not be defined in this step.

REFERENCES

[1] A. Corominas Murtra, J.M. Mirats Tur, and A. Sanfeliu. Efficient active global localization for mobile robots operating in large and cooperative environments. In *IEEE International Conference on Robotics and Automation*, pages 2758–2763, May 2008.

[2] G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, SODA, pages 437–446, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[3] D. Fox, D. Burgard, and S. Thrun. Active Markov localization for mobile robots. *Robotics and Autonomous Systems*, 25:195–207, 1998.

[4] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[5] A. Gasparri, S. Panzieri, F. Pascucci, and G. Ulivi. A hybrid active global localisation algorithm for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 3148 –3153, april 2007.

[6] D. Golovin, A. Krause, and D. Ray. Near-optimal Bayesian active learning with noisy observations. *CoRR*, abs/1010.3091, 2010.

[7] R. He, E. Brunskill, and N. Roy. Puma: Planning under uncertainty with macro-actions. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, Atlanta, GA, 2010.

[8] R. He, E. Brunskill, and N. Roy. Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research*, 40:523–570, 2011.

[9] P. Jensfelt and S. Kristensen. Active global localization for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17(5):748–760, October 2001.

[10] S. Koenig, M. S. B Mitchell, A. Mudgal, and C. Tovey. A near-tight approximation algorithm for the robot localization problem. *SIAM*, 39(2):461–490, 2009.

[11] H. Kurniawati, D. Yanzhu, D. Hsu, and W. S. Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *International Journal of Robotics Research*, 30:308–323, March 2011.

[12] A. Mcgovern. acQuire-macros: An algorithm for automatically learning macro-actions. In *NIPS Workshop on Abstraction and Hierarchy in Reinforcement Learning*, 1998.

[13] J. M. Porta, J. J. Verbeek, and B. J. A. Kröse. Active appearance-based robot localization using stereo vision. *Autonomous Robots*, 18:59–80, January 2005.

[14] M. Rao, G. Dudek, and S. Whitesides. Randomized algorithms for minimum distance localization. *International Journal of Robotics Research*, 26:917–933, September 2007.

[15] C. Tovey and S. Koenig. Gridworlds as testbeds for planning with incomplete information. In *Proceedings of the National Conference on Artificial Intelligence*, pages 819–824, 2000.

[16] C. Tovey and S. Koenig. Localization: Approximation and performance bounds to minimize travel distance. *IEEE Transactions on Robotics*, 26(2):320–330, April 2010.