Continuous Area Sweeping: A Task Definition and Initial Approach

Mazda Ahmadi and Peter Stone
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500, Austin, TX 78712-0233
Email:{mazda,pstone}@cs.utexas.edu
http://www.cs.utexas.edu/~{mazda,pstone}

***Abstract*— As mobile robots become increasingly autonomous over extended periods of time, opportunities arise for their use on repetitive tasks. We define and implement behaviors for a class of such tasks that we call *continuous area sweeping* tasks. A continuous area sweeping task is one in which a robot (or group of robots) must repeatedly visit all points in a fixed area, possibly with non-uniform frequency, as specified by a task-dependent cost function. Examples of problems that need continuous area sweeping are trash removal in a large building and routine surveillance. We present a formulation for this problem and an initial algorithm to address it. The approach is analyzed analytically and is fully implemented and tested, both in simulation and on a physical robot.**

I. Introduction

Consider a robot whose goal it is to keep the floors clean in a large office building. This task requires continual execution: by the time the robot has cleaned the entire building once, some parts have become dirty again. A first-cut approach might lead the robot to simply clean the building from top to bottom and then start over again. However, if the rate at which areas of the building become dirty is non-uniform and possibly even non-stationary, a more sophisticated solution is called for. In particular, the robot should ensure that it cleans highly-trafficked areas, such as the main entrance and the restrooms, much more frequently than, say, the closets.

We define such a task to be an example of *continuous area sweeping* task. A continuous area sweeping task is one in which a robot (or group of robots) must repeatedly visit all points in a fixed area, possibly with non-uniform frequency, as specified by a task-dependent cost function.

Additional examples of continuous area sweeping tasks include trash removal and, the task we consider in this paper, routine surveillance. When performing surveillance, a robot needs to continually traverse its environment in an effort to detect some events of interest, such as gas leaks, water dripping, lights on, open doors, etc. In the surveillance task, a location can be "visited" by observing, rather than by occupying it physically.

The goal of a continuous area sweeping task is not just to sweep the area in the minimum time, but to sweep the area in such a way as to minimize the average event detection time, possibly weighted by the importance of different events. *Event detection time*, is the time-period between event appearance and its detection. Notice that since the event appearance time is not known to the robot, this value is not computable by the robot. The definition of *event importance* is problem-dependent. For example, in the trash collection task, the importance of collecting food trash may be higher than that of collecting paper goods. Minimizing the weighted average event detection time will result in the sensible behavior of visiting kitchens and other public areas more often than (most) individual offices. For the surveillance task, one may define the importance of identifying gas leaks as being higher than finding lights on.

Continuous area sweeping tasks are closely related to the *security sweep* [1], or *sweeping* [2] task. In the security sweep or sweeping task, the robot(s) are to *visit* the whole environment in minimum time. Continuous area sweeping is also related to *coverage path-planning* [3], which "is a new path planning approach that determines a path for a robot to pass over all points in its free space." [3] The relevant differences are that in continuous area sweeping, the sweep must be performed i) repeatedly (continuously), and ii) non-uniformly, that is with more frequent attention given to some areas than to others. As surveyed by Parker [4], most previous approaches to surveillance assume ideal sensors and no computational bounds. In contrast, in this paper we consider solutions that are fully implementable (and implemented) on a physical robot.

We propose a formulation of continuous area sweeping tasks such that the *state* ($s \in S$) is defined as the robot's position and orientation discretized into a grid as well as a representation of how recently each grid cell has been visited. The robot's *actions* ($a \in A$) are defined in terms of the state to which it will navigate (by the shortest possible path) next. The *cost function* is the average time-period between appearance and detection of objects weighted by their importance. The robot's *policy* π is a mapping from states to actions, $\pi : S \mapsto A$. The goal is to find the policy with minimum *cost*, which is computed by the cost function.

We tackle this problem by dividing it into two sub-problems:

1) Learn the rate at which each grid cell accumulates *reward potential*. The *expected reward* of visiting a cell at any given time depends on this rate and the time at which the cell was last visited. **(learning)**

2) Given these expected rewards and knowledge of the robot's (possibly stochastic) transition function, compute a sequence of actions for the robot (*policy*) with minimum cost. **(planning)**

The remainder of the paper is organized as follows. In Section II we formalize the class of continuous area sweeping tasks. Section III introduces an initial algorithmic solution to this class of tasks. In Section IV we instantiate the formalism and algorithms on the robot surveillance domain. Our methods are fully implemented and tested both in simulation and on a physical robot, the Sony AIBO ERS-7 4-legged robot. Section V concludes the paper and discusses future works.

II. Continuous Area Sweeping Formulation

In this section we specify our task of interest in detail. In a continuous area sweeping task, the robot must repeatedly visit all the points in its environment in an effort to detect or react to different types of events $e \in E$. The events can in general have varying degrees of importance, imp_e, and each event may occur in different places with varying frequencies. In the case that all points are equally likely locations for an event of interest, the events are equally important, and the robot needs to be physically present at the point to "visit" it, the problem reduces to the traveling salesman problem. Thus, in general, continuous area sweeping is NP-Hard, and we must rely on approximate solutions.

We begin by dividing the robot's environment into disjoint grid-cells G, with each event occurring in one grid-cell. We consider time as a sequence of discrete steps. The orientation $\theta \in O = \{\textit{North, South, East, West}\}$ of the robot is also considered as being one of 4 disjoint values. We also track the last time a robot has visited each cell $g \in G$ in an array $LV[G]$ by setting $LV[g]$ = current-time whenever the robot visits cell g.

The problem is defined as a tuple $(S, A, T_{sa}, P_{eg}, CF)$, where:

- $S = G \times O \times LV[G]$ is a set of *states*, representing the position and orientation of the robot as well as the array of last-visit times to each cell.
- A is the set of possible actions. The actions in this formulation are specified based on their destinations. In particular, the environment is divided into a *coarse grid* called CG.[1] Each action $a \in A$ is defined as traversing the path between the current position and the center point of one of the coarse grid-cells in CG and at the end turning to reach one of the four orientations. That is, there are $|CG| \times |O|$ possible actions from each state. The time complexity of the algorithm is highly dependent on the number of actions, thus we usually want CG to be coarser than G. Since the map of the environment is assumed to be stationary, the shortest paths between all pairs of points in CG can be computed upon initialization, for instance using the Floyd-Warshall algorithm.
- T_{sa} is the state transition probabilities. Based on the current state and action, it gives the distribution over the states that the robot will transition to. The transition function is stochastic, because based on possible robot localization errors, the robot may end up in grid-cell g_j when aiming for grid-cell g_i.
- P_{eg} is the probability of appearance of event e in cell g per second. For example, if P_{eg} = 0.1, there is the expectation of event e occurring every 10 seconds in cell g.
- CF is the *cost function* of the *policy*. It is not necessarily known to the robot, but is nonetheless used to measure the efficacy of the robot's policy. The cost function that we define for the continuous area sweeping problem is the average time elapsed from appearance to detection of the events, weighted by their importance of the event (imp_e). While the *importance* of the events is known for the robot, the appearance time of each event is not observable to the robot.

The goal of the robot is to find a *policy* $\pi : S \mapsto A$ which minimizes the cost function. The policy determines which action is chosen by the robot in each state.

When the goal is to maximize a reward signal that is observable to the robot, the policy may be learned via reinforcement learning using temporal difference methods [5]. In the next section, we present a heuristic policy that does not rely on environmental feedback.

III. Exploration Algorithm

In this section, we present a detailed description of our initial approach to continuous area sweeping tasks. We begin by assuming that time is discretized into *cycles* representing the times at which the robot can make action decisions. For the purposes of our algorithm, we define an *expected reward* of each grid-cell g at time t as the expected sum of *the event importance values* present in grid g at time t.

The algorithm consists of two main modules:

1) **Learning:** Each cycle the robot updates the learned *expected reward* for all cells $g \in G$. That is, it aims to learn $\sum_e P_{eg} * imp_e$ in each cell g.
2) **Planning:** If the robot is in the middle of performing an action, it continues with that action. Otherwise, using the *expected reward*, it aims to find a *policy* π that minimizes the *estimated cost*, defined as the average expected reward (weighted detection time) of the grid-cells over time. It then executes the first action from the policy. Notice that we are *not* looking for a policy that gains minimum expected reward, but a policy that sustains a minimum expected reward summed over all grid cells. When a cell is visited, the expected reward of that cell is set to zero. Thus an action which visits the cells with highest expected rewards, maximally *decreases* the average expected reward.

The details of these two steps of the algorithm are presented in the Section III-A and III-B. In Section III-C we show

[1] CG need not be related to G in any way, though in general we expect it to be coarser than G.

that this approach approximately minimizes the cost function defined in Section II, which is the goal in the problem formulation.

A. Learning the expected reward

The aim of learning is to approximate the *expected reward* for visiting a cell at any given time. Expected reward is defined as the expected sum of importance values of the events present in grid g at time t. In Section III-C we show how minimizing the estimated cost (average expected reward) will result in minimizing the average *detection time* (i.e. maximizing the policy's value). A greedy approach to minimizing the expected reward is presented in Section III-B.

Formally expected reward is defined as:

$$exp\_reward_{gt} = \sum_{all\ e} (t - LV[g]) \times P_{eg} \times imp_e \qquad (1)$$

Where $LV[g]$ is the last time that cell g has been visited before time t. Notice that the value of $(t - LV[g])$ is known to the robot and is independent of the rest of the equation. Thus, it is only needed to approximate the value of $\sum_{all\ e}(P_{eg} \times imp_e)$. We refer to this quantity as the *potential reward* of cell g. Note that the potential reward of cell g is independent of time: it is the sum of the importance values of the expected events for cell g per second.

The high-level pseudo-code of the algorithm which estimates potential reward for cell g is given in Figure 1. For each grid-cell g, the reward potential pot_reward_g is initialized to $\epsilon > 0$ (in our case $\epsilon = 1$). By initializing the pot_reward_g's to a non-zero constant value, we are assuming that all grid-cells have an equal positive probability of all events occurring. That is, we start with the assumption that $\forall e, g, g', P_{eg} = P_{eg'} > 0$. If we have prior knowledge that some grid-cells have a higher importance event possibility than others, potential reward for those grid-cells can be initialized to a higher value.

```
α = 0.9 (learning rate)
for all grid-cells 'g' do
    initialize pot_reward_g := 1
in each cycle do
    t[g] = current-time - LV[g]
    for each detected event e in grid g do
        pot_reward_g := (1-α)*pot_reward_g + α*imp_e/t[g];
    for each visited g with no event
        pot_reward_g := pot_reward_g * 0.99;
end for
```

Fig. 1. High level pseudo-code for learning the reward potential.

It is assumed that after an appearance of a rewarding event in grid-cell g, the event will remain there until the robot visits g. Thus, if grid-cell g is visited after t_g time-units and the robot visits the events with sum of importance values of *IMP*, it can be assumed that with a higher probability every $\sim t_g$ time-units, an event with the importance value of *IMP* appears in grid-cell g. Whenever a non-empty set of events with sum of *event importance* values of *IMP* are visited in grid-cell g, the following update happens:

$$pot\_reward_g \leftarrow (1 - \alpha) * pot\_reward_g + \alpha * \frac{IMP}{t - LV[g]}$$

Where, α is a learning rate, which in our experiments is set to 0.9. The update rule presented above, changes the estimation of $potential\_reward_g$ to be closer to $\frac{IMP}{t_g}$, which is the assumed sum of events importance values per time for grid-cell g. The estimation of reward potential for g will become more accurate after more visits to grid-cell g.

Since the frequency of event appearance may not be constant over time, there is also a need to unlearn the reward potentials. Thus, every time that the robot visits grid-cell g with no event, it will perform the following update on reward potential pot_reward_g:

$$pot\_reward_g \leftarrow pot\_reward_g * f$$

Where f is an unlearn factor and in our experiments is set to 0.99. This update rule enables the robot to gradually unlearn the one-time events. Notice that the learning of potential reward for grid g happens only when there is an event in g. If no event is detected while visiting grid g, the above unlearning update will be performed. Since a lasting influence of a detected event is desired, The rate of learning is much faster than unlearning

Expected reward is defined as the expected sum of the importance values of the events present in grid g at time t. We compute it incrementally by adding potential reward (expected reward per second) in each cycle. The pseudo code for computing the expected reward for each grid-cell is shown in Figure 2. In each cycle, if a grid-cell is being visited, the expected reward for that grid-cell will be set to zero, otherwise it will be incremented by the amount of the potential reward of that grid-cell. As a result, the expected reward for cell g will be equal to the potential reward of g multiplied by the amount of time that g has not been visited.

```
for all grid-cells 'g' do
    exp_reward_g := 0
for each cycle do
    for all grid-cells 'g' do
        if g is being visited
            exp_reward_g := 0;
        else
            exp_reward_g += pot_reward_g;
```

Fig. 2. High level pseudo-code for computing expected reward for grid-cells in each cycle. The pot_reward is computed in the Figure 1

B. Choosing actions

When choosing an action, the robot can move to the center point of any cell in the coarse grid CG, and after reaching the destination turn to face one of the four orientations $\{$*North, South, East, West*$\}$. We assume that the map of the environment is already known and that the robot has a model of its own (stochastic) motion. As an initial approach, we use a form of greedy action selection.

The pseudo-code to choose the action is given in Figure 3. For each action of going to point cg, the robot computes the trajectory of going to that point. Computing this trajectory is done for all pairs of points at initialization using the Floyd-Warshall algorithm [6]. Each trajectory is divided into discrete points, one point for each cell of G, which is the center of the line segment that passes through G. For each one of the discrete points of the trajectory, the grid-cells that will be seen from that point are computed as follows. We assume a 180-degree field of view for the robot, and the robot computes 181 rays with origin at its position and with angles ranging from -90 to 90 degrees from the robot's orientation. For each one of the lines, the cells that the line passes through before hitting a wall are considered "visited".

The expected reward of these visited grid-cells will be summed up for all the points in the trajectory and the result will be the *expected received reward* of performing the action. After computing the expected reward values, the algorithm greedily choose the action with the maximum expected reward.

The intuition behind this approach is that after the grid-cells with high expected reward are visited, their expected reward is set to zero, thus the estimated cost (average expected reward) decreases. By choosing the action with maximum expected received reward per time, we will have the maximum one-step decrease in the estimated cost. It is possible to use more complex planning approaches to achieve closer to optimal solutions for this formulation, but the greedy approach is sufficient to achieve a good result in this environment.

```
s: state of the robot
A: possible actions in the state s
obs[g]: temp array to avoid double counting
max_reward := minimum_value
for each action a in A do
    a_reward := 0;
    time_a : time to perform a
    for all g do obs[g] := false;
    compute the trajectory T for a
    for each point t in T do
        for each g visited from t do
            if (not obs[g])
                a_reward := a_reward + exp_reward_g;
                obs[g] := true;
            end if
    if a_reward / time_a > max_reward
        max_reward := a_reward/time_a;
        best_action := a;
    end if
end for
perform best_action;
```

Fig. 3. High level pseudo-code for choosing the best action in one cycle.

C. correctness of the approach

In this subsection, we provide a proof that minimizing the estimated cost will result in minimizing the cost function of the problem formulation. For the sake of analysis, we assume a finite horizon, with finite time and events.

The cost function in the formulation is the average detection time multiplied by the importance of the event. The goal is to minimize the cost function:

$$minimize(\sum_{e=1}^{E}(detect\_time_e \times imp_e)) \qquad (2)$$

Where E is the number of events in our finite horizon, $detect\_time_e$ is the detection time of event e and imp_e is the importance value of event e.

The goal in the presented approach is to minimize the estimated cost over time. That is:

$$minimize((\sum_{t=1}^{C}\sum_{g=1}^{|G|} exp\_reward_{gt})) \qquad (3)$$

Where, $|G|$ is the number of grids, C is the number of cycles in our horizon and exp_reward_{gt} is the expected reward of grid-cell g at time t.

By the definition of expected reward (Eq. 1), in the finite horizon we have:

$$exp\_reward_{gt} = \sum_{e=1}^{E}(t - LV[g]) \times P_{eg} \times imp_e \qquad (4)$$

Where $LV[g]$ is the last time that grid-cell g has been visited before time t and P_{eg} is the probability of appearance of event e in grid-cell g.

Based on equations 3 and 4, the goal of the proposed approach is to minimize the following equation:

$$\sum_{e=1}^{E}\sum_{t=1}^{C}\sum_{g=1}^{|G|}((t - LV[g]) \times P_{eg} \times imp_e) \qquad (5)$$

The average value of $(t - LV[g])$ over time (average detection time) is equal to $0.5T_g$, where T_g is the average time between two visits of the robot to grid-cell g. Thus minimizing the Eq. 5 results in minimizing this equation:

$$\sum_{e=1}^{E}\sum_{g=1}^{|G|}(T_g \times P_{eg} \times imp_e) \qquad (6)$$

Notice $\sum_{g=1}^{|G|}(T_g \times P_{eg})$ is the *expected detection time* of grid-cell g and since imp_e is independent of g, minimizing the above equation will result in minimizing the cost function (Eq. 2).

In this section, we showed that by achieving the goal of the proposed approach (i.e. minimizing the estimated cost over time) the cost function of the problem formulation will be minimized (which is the goal of the optimal policy). We are using a greedy approach to minimize the estimated cost over time, which is not necessarily optimal, but given the proven fact that minimizing estimated cost will result in minimizing the cost function, it is a reasonable approach.

IV. Experimental Results

To test our approach, we have implemented and evaluated our algorithm on a physical robot in a representation of the routine surveillance task. As our robot, we use a Sony ERS-7 four-legged AIBO robot (Figure 4). The robot's sensor device for "visiting" locations in its environment is a camera mounted on the head of the robot. It can capture 208 $\times$ 160 frames of pixels at roughly 30Hz. Due to the computational intensity of image processing, our robots typically make decisions at roughly 25Hz, thus the cycle defined in Section III is set to 0.04 second. By turning its head, the robot can gain a 180-degree field of view. It has 20 degrees of freedom and a 576Mhz on-board processor.

Fig. 4. ERS-7 Sony AIBO robot

As baseline software, we use the UT Austin Villa code base [7], which provides robust color-based vision, fast locomotion, and reasonably accurate localization within a 2.9m $\times$ 4.4m area[2] via a particle filtering approach. Even so, the robot is not, in general, perfectly localized, as a result of both noisy sensations and noisy actions. The robot also has limited processing power, which limits the algorithms that can be designed for it. G is equal to a 18 $\times$ 15 grid, that is we discretize the robot's environment into an 18 $\times$ 15 grid. CG, which defines the available actions, is set to a 6 $\times$ 5 grid. There is just one type of event in the environment, which is the appearance of an orange ball that the robot can recognize from anywhere on the field provided that it has an unobstructed view. We test two different configurations of the world with the real robot. One other configuration is tested in a custom-built AIBO simulator [7]. The simulator, though abstract with respect to locomotion, provides a reasonable representation of the Aibo's visual and localization capabilities, and allows for a more through experimentation, particularly with regards to testing different distributions of ball appearances.

A. *Configuration I with real robots*

As an initial experiment, we configured the robot's environment as shown in Figure 5. A picture of the actual environment with the robot is shown in Figure 6. The robot knows the locations of the walls in the environment, but must decide for itself how to move so as to perform surveillance.

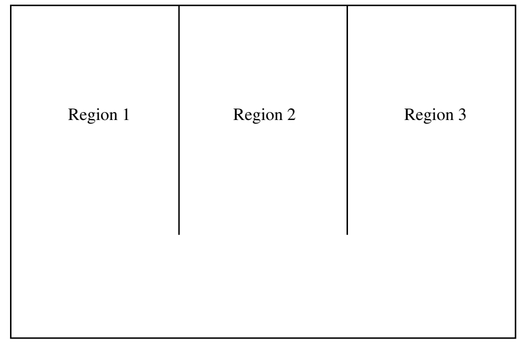

Fig. 5. Representation of configuration I

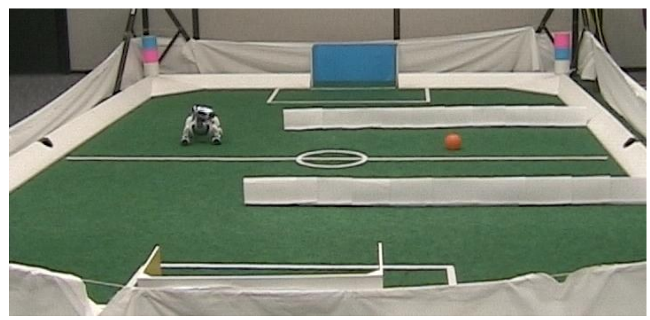

Fig. 6. Picture of configuration I with the real robot

Before appearance of the balls, the path that the robot found is *path 1* in Figure 7. It is the minimal path for uniformly visiting the whole environment. After that, we started to show the balls to the robot in region 1 and 2, but not region 3. In this new situation the robot found *path 2* in Figure 7. By traversing this path, robot visits region 1 and 2 more often than 3 and that is a desirable result.[3]

Later in the experiment we stopped showing any balls to the robot. As a result of the forgetting parameter, the robot gradually went back to uniform exploration (*path 1* in Figure 7). Finally, we again started to show the ball, this time in regions 2 and 3. The path that the robot found in this new situation is *path 3* in Figure 7.[4]

B. *Configuration II with real robots*

As a follow-up to this initial experiment, we created a more complex environment as is illustrated in Figure 8 and pictured

[2]The field is as specified in the 2004 rules of the RoboCup Four-Legged Robot League: `http://www.tzi.de/4legged`

[3]Videos of the robot in action in this environment are available from `http://www.cs.utexas.edu/~AustinVilla/?p=research/surveillance`, named configuration I part 1

[4]A video of the robot forgetting what it has learned before and re-learning the new distribution of the ball appearances is available at `http://www.cs.utexas.edu/~AustinVilla/?p=research/surveillance`, named configuration I part 2

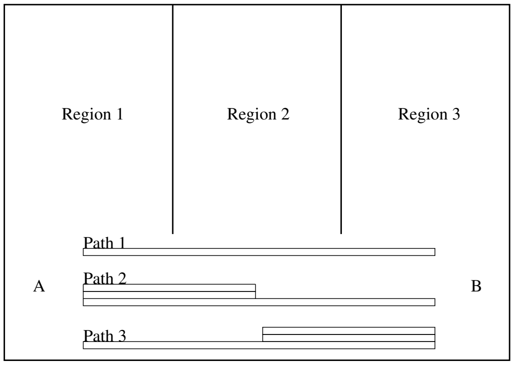

Fig. 7. The path the robot traverses in configuration I

in Figure 9.

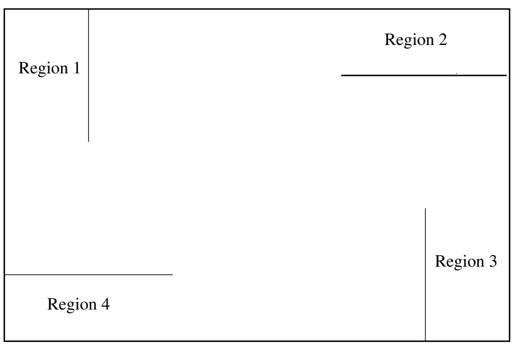

Fig. 8. Representation of configuration II

Before the appearance of the balls, the path that the robot finds is very similar to the path shown in Figure 10, which is the minimal path for the uniform appearance of the balls. Because of the noise in the environment, the robot does not exactly follow that path, but the path is very close to that one. After showing the ball for several times in regions 1 and 3, the path that the robot finds is close to the one shown in Figure 11. [5] As is apparent from the path, because of the higher probability of appearance of the balls in *region 1* and *3*, those regions are visited more frequently.

The robot experiments verify that our approach can run in real time on a computationally limited platform, and that the robot can operate using the same algorithm in multiple environments. The robot is made aware of the locations of the walls, but given that information is able to recompute its actions from scratch. However, due to the time-consuming nature of running experiments in the real world, we further validate our approach in simulation.

[5]A video of this experiment is available at `http://www.cs.utexas.edu/~AustinVilla/?p=research/surveillance`, named configuration II

Fig. 9. Configuration II with real robot

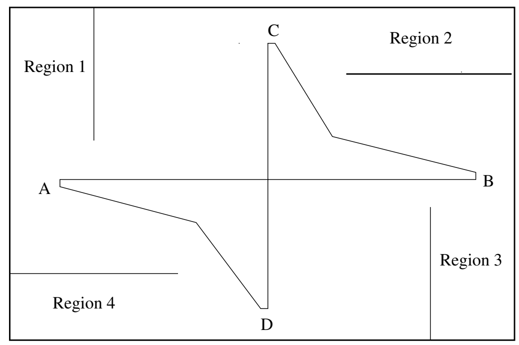

Fig. 10. The path that the robot traverses in uniform distribution of the appearance of the ball.

C. Simulation

We use our custom-built AIBO simulator to test our approach. The architecture of our AIBO code is designed in two layers. The lower layer is responsible for managing the visual sensor and translating high-level commands to robot motor commands, while the upper layer reasons about the visual inputs and issues high-level motion commands. Our simulator is designed to take the place of the environment and the lower layer, providing abstract visual input to the control code, and simulating robot motions. Full details of the code organization and simulator interface can be found in our technical report [7].

With this simulator platform, we are able to run identical upper-level code both on the robot and on the simulator. Using the simulator, we are able to control precisely the distribution of locations at which the ball appears, and we are able to run experiments much more quickly.

In our simulation experiments, we test four different distributions of ball appearances in configuration II from the real robot experiments (Figure 8). In all the distributions, the ball appears in each cell g with probability $P_{eg} * 50$ every 50 seconds. In particular, this means that a ball can appear in more than one cell before the robot sees any of them, and that there can be more than one ball in the same place by the time the robot visits that place. P_{eg} is different in each one of the experiments.

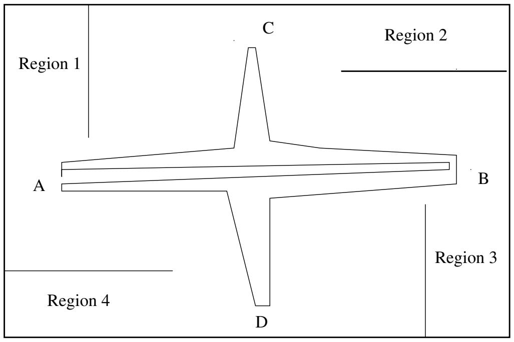

Fig. 11. The path that the robot finds when there is several appearances of the ball in region 1 and 3

The approximate time needed (in seconds) for walking between each pair of points is given in Table I, as measured on the physical robot. Notice that the time it takes to walk straight across the field — even the long way from A to B — is less than the time it takes to move between adjacent points such as B and D. The reason for the difference is that the robot takes significant time to turn.

Point 1	Point 2	Time needed to traverse
A	B	50
A	C	55
A	D	55
B	C	55
B	D	55
C	D	40

TABLE I
APPROXIMATE TIME IN SECONDS NEEDED TO WALK BETWEEN EACH PAIR OF POINTS IN CONFIGURATION II.

In the following subsections, we discuss the results in each of four distributions: a uniform distribution; a distribution in which the ball only appears in one place; a biased distribution; and a non-stationary distribution.

1) Uniform Distribution: In our first experiment, the ball appears with identical probability in each of the four regions. Notice that were the ball to appear anywhere else in the environment, the optimal policy would not be affected significantly, since almost all of the actions visit the center part of the environment. The path that the robot finds after learning is approximately the path shown in figure Figure 10.

In order for the robot to visit all the regions it at least needs to go through the points A, B, C and D or a small area around them. Based on the travel times in Table I, the solution shown in Figure 10 is optimal, which we can check by exhaustive search.

While following the path that the robot found, each ball was approximately visited after 106 ± 2.1 seconds. In the experiment, 100 balls were shown to the robot. The optimum average detection time is 100 seconds, which is the result of traversing the path in Figure 10. The whole traversing time for that path is $55 + 40 + 55 + 50 = 200$. Thus, it takes between 0 and 200 seconds to detect each event, and the average detection time is 100 seconds. Thus, our approach is in 4% margin of error, which is quite close to optimal, with the error coming mainly from action noise. The path that the robot found is not exactly the one in Figure 10, since for example the robot does not always go to the exact grid-cell of point A. Rather, to visit region 1, it goes to one of the grid-cells close to A. The exact motion of the robot is visible in our on-line videos.

This initial experiment verifies that our greedy algorithm can produce the optimal solution in the most benign case.

2) Always in One Region Distribution: As a second test, we created a distribution such that the ball only appeared in region 2. The path that the robot finds is approximately the one shown in Figure 12. It took the robot only one pass through the field to approximately follow the path in Figure 12.

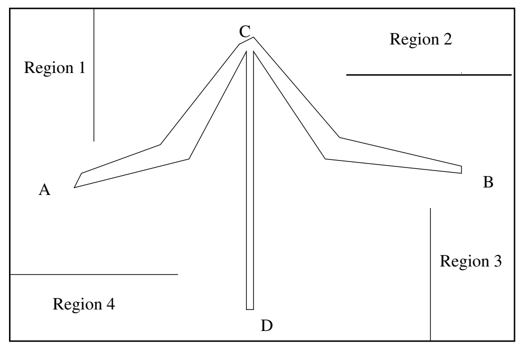

Fig. 12. The path the robot traverses when the ball always appears in region 2.

On average, every ball is noticed 47 ± 1.3 seconds after its appearance. In the experiment 100 balls were shown to the robot. Of course, the optimum solution here is for the robot to stay close to region 2 and visit the balls right after it appears, but our algorithm enforces the constraint that the robot should continually visit the whole environment at least periodically in case new events occur. With the condition that we want to visit the whole environment *continually*, 47 seconds is reasonable. In particular, learning the distribution has gained us a 50% performance improvement: without learning the robot would traverse the environment uniformly, resulting in an average detection time of 106 seconds or ideally 100 seconds. Notice that the average detection time in uniformly traversing the environment and visiting each cell once in the full traverse of the environment is independent of the distribution of event appearances. Because whatever the distribution, the average detection time for each event is constant and equal to the half of the whole traversing time.

3) Biased Distribution: In our next experiment we tested the robustness of our approach in a scenario such that the balls appear in all the regions but with different probabilities. In the biased distribution, with probability 60% the ball appears in region 2, with probability 30% it appears in region 1, with probability 5% in 3, and with probability 5% in 4 (P_{eg} =

.6/50, .3/50, .05/50, and .05/50 respectively). The path that the robot traverses is approximately the one in Figure 13.

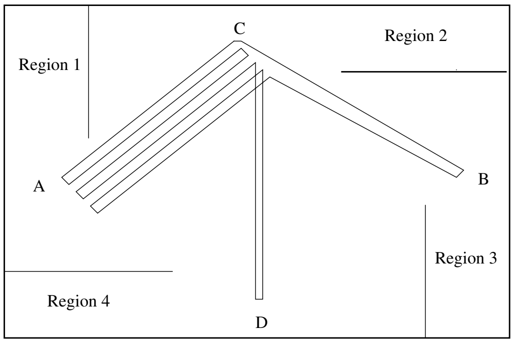

Fig. 13. The path that the robot traverses in the face of a biased distribution, where the chance of the ball appearance is 60% in region 2, 30% in region 1 and 5% in regions 3 and 4.

The time needed for the robot to change its path from the uniform case to the path shown in Figure 13 is based on how fast it can learn the distribution, which itself is based on frequency of ball appearances. In our experiment, every 50 seconds an average of 1 ball appeared. In this setting, the robot took 9 complete traverses (1734 seconds or 35 ball appearances) to start traversing the shown path.

After the 1734 seconds, when the robot learned the distribution, on average every ball was visited 79 ± 1.2 seconds after its appearance. In the experiment, 200 balls were shown to the robot. This result is significantly better than uniform traversal which results in average detection time of 106 or ideally 100 seconds.

4) Changing Distributions: In our final experiment we tested the robustness of the approach to changing distributions. In particular, we consider a scenario in which at some unknown point in time the probability of appearance of the balls changes abruptly. The initial distribution of the ball appearance was the same as the biased case discussed in previous section, that is 60% in region 2, 30% in region 1, 5% in region 3, and 5% in 4. After 100 ball appearances, the distribution changes to the uniform appearance of the ball.

The path that the robot found with the starting distribution is the same as the one in Figure 13. It took the robot about 1820 seconds or 36 ball appearances to adapt to the second distribution and approximately follow the path in Figure 10.

V. Conclusion and Future Works

In this paper, the problem of *continuous area sweeping* is introduced. The problem is defined as one in which a robot must repeatedly visit every part of the environment in order to detect a set of events of interest. The frequency of the events can possibly be non-uniform, thus the robot should visit the points with non-uniform frequency. Examples of continuous area sweeping tasks are surveillance and cleaning.

In this paper, we formalize the problem and introduce an initial approach that non-uniformly visits the environment to minimize the estimated cost. The approach is analyzed analytically and is tested both in simulation and on real robots.

Our on-going research agenda includes expanding the robot behavior to include non-greedy planning and cooperative multi-robot interactions.

Although there is a good deal of uncertainty in the object recognition on the Aibo, we currently do not express it explicitly other than by folding it into the estimate P_{eg}. Note that we *do* represent state transition uncertainty within the function T. Representing object detection noise explicitly is also a direction for future work.

Acknowledgment

The authors would like to thank the members of the UT Austin Villa team for their efforts in developing the software used as a basis for the work reported in this paper. Special thanks to Greg Kuhlmann for developing the simulator. This research was supported in part by NSF CAREER award IIS-0237699 and ONR YIP award N00014-04-1-0545.

References

[1] N. Kalra, A. T. Stentz, and D. Ferguson, "Hoplites: A market framework for complex tight coordination in multi-agent teams," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-04-41, August 2004.

[2] J. A. T. Y. E. Kurabayashi, D. Ota, "Cooperative sweeping by multiple mobile robots," in *Proc. of IEEE International Conference on Robotics & Automation (ICRA)*, 1996.

[3] H. Choset, "Coverage for robotics; a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.

[4] L. E. Parker, "Distributed algorithms for multi-robot observation of multiple moving targets," *Autonomous Robots*, vol. 12, no. 3, pp. 231–255, 2002.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[7] P. Stone, K. Dresner, P. Fidelman, N. K. Jong, N. Kohl, G. Kuhlmann, M. Sridharan, and D. Stronger, "The UT Austin Villa 2004 RoboCup four-legged team: Coming of age," The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, Tech. Rep. UT-AI-TR-04-313, October 2004.