Coverage Path Planning for Mobile Cleaning Robots

Marten Waanders
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
m.j.c.waanders@student.utwente.nl

ABSTRACT
In this paper a path planning algorithms for a mobile cleaning robot is developed. An exact cellular decomposition divides an environment in multiple smaller regions called cells. To produce an efficient cleaning path, the order in which these cells are cleaned is important. This paper describes how the optimalization of this path can be formulated as a Traveling Salesman Problem. Furthermore an exact cellular decomposition called the Boustrophedon cellular decomposition is expanded to handle the dynamic, unknown environments a cleaning robots can face in an indoor environment. In particular this is achieved by allowing the robot to change the decomposition during its cleaning activity. A simulation study have been used to validate this approach.

KEYWORDS
Coverage path planning, cleaning robots, exact cellular decomposition

1. INTRODUCTION
Cleaning robots are robots that are designed to clean indoor environments autonomously. They are service robots that currently have a great market potential unlike most of the other service robots [13]. Since they are currently commercially attractive, research is being done into making improved versions of these robots. One of the areas in which these robots can be improved is in the path they take for cleaning the environment.

Older cleaning robots usually featured simple behaviours made by a random path scheduling algorithm, which is sometimes combined with programmed trajectory patterns such as spirals and serpentines and a wall following mechanism. Those that had a better algorithm for producing their path were a lot more expensive than their randomly moving competitors [13]. In [12] the coverage of some simple commercially available robots was measured. The maximum area covered turned out to be dependent on the robots shape: "The maximum area covered is highly correlated with how close the robot can go to the walls and the radial distance from the cleaning device to the external structure of the mobile robot" [e, page 6].

Coverage path planning is a path planning approach that determines a path that includes every reachable point in the target environment [5]. To make an efficient path through its environment, a cleaning robot therefore needs to use a good coverage planning algorithm.

Many kinds of coverage algorithms have currently been devised. Approaches to coverage path planning include genetic algorithms, neural networks, exact cell decomposition, spanning trees and spiral filling paths [4,t], template-based models and artificial potential fields. In section 2 some existing solutions for coverage path planning are mentioned along with their advantages and disadvantages.

Most coverage algorithms that ensure complete coverage do so by creating a cellular decomposition of the area. A cellular decomposition breaks up the environment into a number of disjoint cells whose union is the entire environment. These cells are created such that each of them can be easily covered [5]. We will discuss cellular decomposition in more detail in section 2. To create such a decomposition a map of the environment is useful. Some algorithms assume the availability of a map while others construct one while cleaning the environment.

One of the complexities in coverage path planning for a cleaning robot is that the environment is generally not static. During the cleaning process doors may open and close, obstacles may be repositioned and people and pets can move around freely. Improving path planning algorithms to deal with more and more complex situations is one of the areas in which research is being done. In this paper a coverage path planning algorithm will be developed that is designed for the use in a cleaning robot which takes dynamic events that are likely to occur in an indoor situation into considerations. This will be done without the assumption that a map is always readily available.

To be more specific we consider a cleaning robot with the following specification;

The robot has a circular shape and has a fixed maximum speed. The robot is capable of moving in straight lines and can turn while staying at the same location. Since some of the current randomly moving cleaning robots also meet these specifications [12], the technology for creating such a robot is readily available. For calculating the covered area we assume that the robot can clean a circular area around its centre. We call this radius the cleaning radius. In this paper the robots cleaning radius is set to 80% of its radius.

The robot can receive sensory data by using an infrared range finder. The robot is able to rotate the infrared range finder with a constant velocity relative to its torso.

For the context of cleaning robots the following requirements arise;

The cleaning robot has to achieve an area coverage that is very close to the maximum coverage. (The cleaning robot should achieve the maximum area coverage attainable by that robot in a static known environment. In a dynamical environment it

15th Twente Student Conference on IT, June 20st, 2011, Enschede, The Netherlands.

should continue to clean uncovered areas until it can no longer deduce the existence of areas that are not already covered.)

Other than coverage, time-to-completion is another important criterion for the robots usability. Cleaning robots usually work on batteries making a long time-to-completion unattractive. Making less turns is also better, since turns cost more time and energy than moving in a straight line and also increase dead-reckoning errors (dead-reckoning is the process of estimating the current position based on a previously determined position, and on the known or estimated speed, elapsed time, and course) [5]. The cleaning robot therefore also needs to minimize the amount of turns it takes.

An apriori map is not necessarily available. Therefore the robot needs to be able to function without needing a map of its environment as input.

The cleaning environment can change between and during cleaning sessions. The robot needs to update its map and cleaning path accordingly.

Although no explicit limit is placed on the robots computational power, in order to be commercially feasible the robot should not need to do a large amount of computation and should not need to use a large amount of memory. The algorithms it uses therefore needs to be efficient.

Due to the limited information the robot can gain at a single time instance the robot needs to create a map based on its sensory data and dead-reckoning. Any inaccuracies in sensor information and dead/reckoning can accumulate (because of the use of dead-reckoning) causing the coverage and efficiency of the path taken by the robot to deteriorate over time. The robot should try to minimize the effect of these errors.

This leads us to the following question: "How can a decent coverage path planning algorithm be made for the specified autonomous cleaning robot taking the above requirements into account?"

And its sub questions:

- Which coverage path planning algorithms exist already and which strengths and weaknesses do these algorithms have?
- How does the newly developed algorithm compare to those already available in coverage, speed and handling moving objects?

To answer these questions some the currently available were investigated and analyzed. The created algorithm was compared to a random path planning algorithm to evaluate its performance. This paper is organized as follows. In section 2 we will give an overview of some of the coverage path planning algorithms that are currently available. In section 3 we explain the algorithm we have developed for our cleaning robot. The simulation studies of the proposed algorithm are presented in section 4 and in section 5 our results will be summarized.

2. EXISTING SOLUTIONS

The simplest case of coverage planning is covering an empty room. Two standard ways of covering an empty room are using a spiral pattern and using back and forth-motions. Both of these patterns can be a basis for an coverage algorithm that works in general environments. These basic patterns are shown in figure 1. In these figures the path made is indicated in black lines. The robot is drawn at its final position to indicate its size in comparison with the environment. The covered area is indicated in grey. Area left uncovered is coloured white. A third common way of covering a simple room is by making use a random path. Its main advantage is that it works on any room regardless of its shape. It is also the simplest algorithm, which is why we discuss it first.

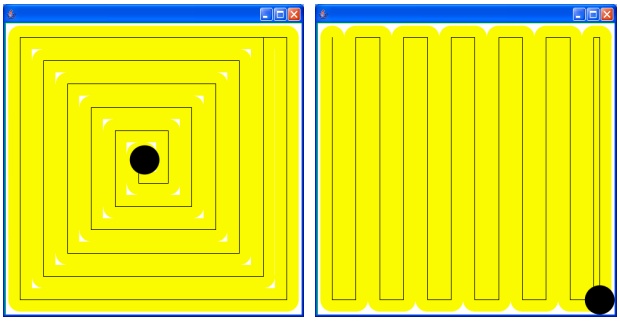

Figure 1. Spiral (left), Back and forth motion (right)

2.1 Random path planning

A simple way of covering the entire terrain is to make the robot move in an arbitrary direction until the current path is obstructed. Once the robot can not continue in its current direction this procedure is repeated. This is a random path planning algorithm. It is sometimes combined with programmed trajectory patterns such as spirals and serpentines and/or a wall following mechanism to increase its efficiency. It has been proven that as the cleaning time goes to infinity a random path algorithm will eventually reach the maximum coverage [12].

Random path planning has its advantages: It does necessitate a lot of computations to calculate a path, and few sensors are needed. Additionally the robot does not need to use a map of its environment. Another large advantage of random path planning is that the algorithm can reach maximal coverage in any environment, regardless of the shape, even if the environment is dynamically changing and unknown. It is however inefficient as the path can have considerable overlap with itself [1] and it does not give any guarantee how much of the area has been covered at the termination time. According to [15] users of cleaning robots have a higher satisfaction with their cleaning robots when they need to do less additional cleaning themselves. Therefore it is important that cleaning robots can clean the entire room. Therefore having a complete path coverage algorithm, instead of a random one, can be an important property for a cleaning robot.

Other disadvantages of random path planning are that the algorithm can take a long time to get out of certain areas (see for instance figure 5 in [1]), and a large variation in the covered area between different cleanings [1].

2.2 Exact cellular decomposition

As already mentioned cellular decomposition breaks down the target environment into disjoint cells which are easy to cover. The coverage can be proved complete if the algorithm ensures that every cell in the decomposition will be visited [r,y].

An advantage of exact cellular decomposition is that it subdivides a complicated problem (finding a path that covers the entire environment) into a couple of simpler problems. Namely how to split up the environment into cells, how to cover each individual cell, how to efficiently move from one cell to another and how to efficiently move from cell to cell. Solving all these simpler problems is generally easier than covering the single complicated problem. Each individual problem can be handled separately and optimizing the path taken can be done by optimizing the solutions to each of these individual sub problems.

Other than providing complete coverage exact cellular decomposition has the advantage that the decomposition does not depend heavily on the dynamics of the robot. The algorithm for covering single cells may need to be adapted, but the way the decomposition is constructed can usually stay the same

Another category of cellular decompositions is the approximate cellular decomposition, which splits up the environment using a grid. Every cell is then a single square on the grid, which is considered covered as soon as the robot reaches it. In exact cellular decompositions these cells can in principle have any shape as long as the cells are disjoint with their union being the entire environment. The shape of the cells is therefore chosen to suit the algorithm used to cover a single cell.

A general disadvantage of exact cellular decomposition is that moving from cell to cell can require the path taken to overlap itself. Additionally it requires a path planning algorithm to find efficient paths to travel from one cell to another.

2.3 Boustrophedon cellular decomposition

The Boustrophedon Cellular Decomposition is an exact cellular decompositions where each cell is created such that it can be covered with a simple back and forward motion pattern as shown in figure 1. (The trapezoidal decomposition [6] and MSA decomposition [10] also make a decomposition suitable for back and forth motions.)

In figure 2 a typical cell the Boustrophedon Cellular Decomposition creates is shown. After taking a specific direction in which the robot will do the back and forth motions the cell will have two boundary edges parallel to this direction, which are labelled A and B in figure 2. A sweep line can be defined as any line parallel to this A. The other boundaries of the cell (labelled C and D in figure 2) can have an arbitrary shape as long as the intersection of a sweep line with C (or D) is a line or a point. The Boustrophedon Cellular Decomposition keeps the direction of the sweep line the same for each cell. This gives sufficient constraints such that the decomposition can be made efficiently.

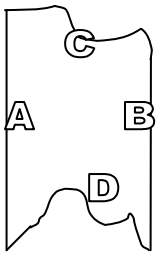

Figure 2. A single cell

Do note that the Boustrophedon Cellular Decomposition does not necessitate the use of any particular algorithm to clean the cells it creates. The created cells can be covered with any coverage algorithm including random motion path planning. However the cells are created such that back and forth motion in the predefined direction is a simple way to cover them. In the next paragraph we will describe an algorithm that uses a spiral motions. Even with seemingly simple boundaries C and D, the spiral motion can be an inefficient way to cover the cell. See for instance Figure 4 which gives a cell that can be covered using back and forth motion with a horizontal sweep line. Spiral filling needs a more turns to cover the cell. Additionally when needing to move to the next cell it can be preferred to begin and end around a boundary of a cell instead of near its center.

In general which cell coverage algorithm works performs best depends a lot on the cell to be cleaned. However, back and forth motions are the default way of covering the cells, since they are simple and tend to be decently efficient for most cells. A simple way to optimize covering individual cells is to consider multiple algorithms for each cell and (heuristically) choosing the one that suits that cell best. In an unknown environment, the path taken to cover a cell usually has some deliberate overlap with itself to prevent small measurement errors from giving bad coverage.

Once a sweep line direction has been chosen the Boustrophedon Cellular Decomposition can be constructed. To do so the obstacles are first extended such that when the robots centre coincides with the boundary of an extended obstacle, the robot will be adjacent to the original obstacle.

The Boustrophedon Cellular Decomposition is constructed by taking the sweep line and moving it orthogonally to its own direction across the environment starting from one edge of the map until it is at the other edge. Everywhere where the connectivity of that line changes, new cells are created. Please refer to [1] for the details of this approach.

In this section we will only give a brief description of the simpler algorithm for creating cells discussed in [6]. For this implementation every obstacle is approximated by a polygonal and all obstacles are assumed to be convex and disjoint. Under these conditions, for each obstacle, the cell in which it lays will split into two new cells at the point of its boundary with the smallest x value (This point is called an IN event). Similarly the cells above and below the obstacle will be merged at the point with the largest x (which is called an OUT event). After doing this for each obstacle a Boustrophedon Cellular Decomposition is made. All cells will now have the form of figure 2 and can be represented by a list of edges making up their upper boundary together with a list of edges making up its lower boundary (specifying lines A and B would be redundant). In figure 3 the black lines give a Boustrophedon cellular decomposition for that environment. Handling the path between cells can be done by a traditional path planning algorithm.

Now all that remains to be discussed is an algorithm that ensures that each cell is covered once. To do so a heuristic solution to the travelling salesman problem can be used. In section 4.1.1 we will consider this in more detail.

In [10] an optimalization to the Boustrophedon cellular decomposition is made. The researchers try to improve the decomposition by taking the optimal angle in which to make the back and forth motion. Their optimalization criteria is to attain a decomposition requiring a minimal number of turns to cover each cells. This is done by using the knowledge that the amount of turns taken has a high correlation with the length of the cells created in the direction orthogonal to the sweep direction. They therefore minimize those lengths to get a good approximate result. Their result is that the optimal sweep direction to take for the Boustrophedon cellular decomposition is one perpendicular to one of the edges of the environment. (Their MSA decomposition, which creates cells taking back and forth motions in multiple directions into account, generally creates a better decomposition than the Boustrophedon cellular decomposition but is computationally expensive.)

An additional disadvantage of Boustrophedon cellular decomposition is that back and forth motions usually leave small uncovered areas around the boundary of every obstacle. In figure 3 an indication of the uncovered area is given. In [6] the author indicates that this can be prevented by following the contour of every obstacle after the original cleaning done by the Boustrophedon cellular decomposition. This however decreases the efficiency of the path.

Although the Boustrophedon cellular decomposition described in [6] is for polygonal known environments, in [1] a version is described that works for unknown environments that need not be polygonal. A major advantage of this algorithm is that it requires the robot to store very little information about the environment while still ensuring complete coverage.

2.4 Backtracking spiral algorithm (BSA)

Spiral filling paths cover the area starting from the outside and going towards the center. In figure 1 a typical spiral pattern is shown. In practically every environment a single spiral can not cover the entire area and thus multiple spirals need to be made, requiring a procedure like backtracking (going back to places where the robot has been before, but where it found a way to go to a still uncovered area) to ensure complete coverage. [8]

The backtracking spiral algorithm mentioned in [8] is an example of an approximate cellular decomposition. In [9] the algorithm is improved to also handle the covering of cells within this grid that are partially covered by an obstacle (BSA uses a coarse-grain grid).

A disadvantage of the backtracking spiral algorithm is that it needs to store all the locations it has visited. Therefore extending the algorithm to one that does not require a grid representation, does not directly allow for an improvement in the amount of memory needed to be stored by the robot. By sticking to a grid representation the algorithm can be used in online coverage, since it only uses local data and data about the path it already covered, as long as the robot is placed at an edge of the room it needs to cover.

The backtracking spiral algorithm also has the disadvantage that small obstacles can cause a path that contains a lot more turns than simple back and forth motions would need, see for example figure 4 (for clarity of the figure the gird is not drawn explicitly).

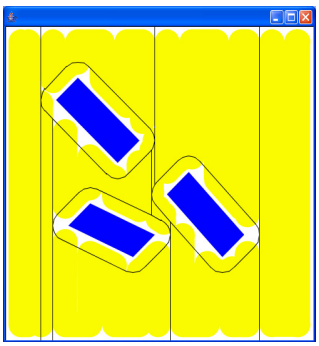

Figure 3.Bad coverage around the edges of obstacles

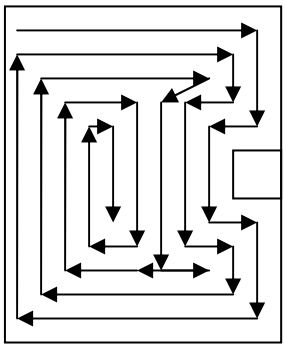

Figure 4. A small object can have a large effect on the number of turns taken by the BSA algorithm.

Although the backtracking spiral algorithm tends to have a lot less overlapping paths than random path planning algorithms it is not free of overlap either. This algorithm has overlap every time it needs to backtracks to create a new spiral. In the example of figure 4 the algorithms needs to backtrack three times. The total overlap in this example is minimal because the distance between the place a spiral ends and the nearest location a new spiral can be planned is small. In larger environments this need not be the case.

Finally it has a disadvantage in getting maximum coverage. If the straight lines are placed a maximum distance apart from each other (to improve efficiency) the corners of the spiral motion will have small uncovered areas as can be seen in figure 1. To avoid this the path needs to have a certain amount of overlap with itself even when backtracking costs and pose errors are not considered.

2.5 Neural network

In a neural network approach the area is represented by points (neurons) and every point is given a value that is dependent on its state (being an obstacle/clean/unclean) and on the value of the other neurons. It is an approximate cellular decomposition where the neurons represent the cells. In [14] this update is only dependent on the neurons that are in a specific neuron's proximity, and the update also depend on the direction the robot is going in order to avoid a large number of turns in the generated path. These values are continuously updated and the robot tends to go in the directions of the neurons with the highest values [15]. The advantage of the approach used in [14] is that it can handle dynamic changes to the environment without deadlocking, while still covering the entire environment. However in [14] the environment is considered to be completely known at all times. The path this approach creates is dependant on the parameters defined on the neural network. In [14] the chosen parameters caused the path taken to resemble one that can be made by using the Boustrophedon cellular decomposition with back and forth motions for covering the cells. However other motion patterns like spirals can also be created with this approach. Therefore this approach has the advantage that it can automatically generate paths that other coverage algorithms create, simply by changing the model parameters appropriately. The downside of this approach is that the size of the neural network can be large depending on the amount of accuracy that is needed, increasing the amount of computations needed to be done. Furthermore, since Neural networks use a grid map they have all the other corresponding problems (BSA uses a coarse grain grid map, reducing these problems a bit). In [11] these some of these disadvantages are given in the following text: "However, grid maps impose significant memory and computational requirements that often make them unsuitable for use in inexpensive robots. Furthermore, the "convergence" of an occupancy grid map relies on obtaining many observations of each grid cell. While this is reasonable with high-resolution sensors such as laser rangefinders, it is more difficult for robots equipped with simpler sensors such as infrared rangefinder arrays."

3. EXTENDING A CELLULAR DECOMPOSTION APPROACH

In this section we propose a coverage algorithm for the use in the cleaning robot specified in section 1. This algorithm is an extension of the Boustrophedon cellular decomposition as discussed in section 2. In this paper we will extend the Boustrophedon cellular decomposition to handle dynamic environments without needing complete knowledge about the environment. Our adaptations can however be applied to other exact cellular decompositions as well. We will create a map of the environment and will make a cellular decomposition based on that map. To handle dynamical changes and unknown areas the cellular decomposition the robot creates will no longer be static. Instead it will be adapted whenever new knowledge about the environment is available.

An exact cellular decomposition approach is used because larger complicated environments tend be harder to cover than a collection of smaller simpler created by a decomposition. One reason for this is that large environments require more bookkeeping. To ensure coverage the robot needs to know which areas it has already covered and depending on the specific algorithm being used the data that needs to be stored in larger environments can be considerable (e.g. neural network approaches can have this problem). When subdividing the environment in smaller areas, we only need to remember which cells still need to be cleaned, which can be significantly less data than remembering the exact shape of the unclean area. (In a dynamically changing environment however some additional

care must be taken if the coverage of the robot is to be maximized.)

In [1] a cellular decomposition is made without storing it. Instead it is represented by a small amount of points. We want to store the complete map of the environment, because we want take dynamic changes into account, which can not be detected as easily if we only store the locations of a limited amount of points.

To do so SLAM (simultaneous localisation and mapping) needs to be applied. This paper will not go into the details on how to apply SLAM to robots with limited sensing abilities. In this paper we assume the availability of a appropriate SLAM algorithm that maps the environment. We will however give some attention to how the robot can plan its path with limited knowledge about its surroundings.

3.1 Static known environment

We first describe how our algorithms works for a known static environments. We define the outer boundary of a room as the outside boundary of the space in which the robot can stand. The following steps are a basic outline of the algorithm.

Step 1: The robot moves to the outer boundary of the environment.

Step 2: The robot follows this boundary until it is has completely circled around it.

Step 3: A Boustrophedon cellular decomposition of the environment is created.

Step 4: Using the cellular decomposition devised in step 3 we create a list containing every cell. The list starts with the cell where the robot is currently located.

Step 5: The cells in the list created in step 4 are covered in sequential order.

For step 3 we use the results of [10] to determine a good sweep line direction and use it for creating the Boustrophedon cellular decomposition of the environment. Thus we try each sweep line direction perpendicular to one of the edges of the environment and take the one resulting in a decomposition where the cells have the smallest total width (orthogonal to the sweep direction). If the amount of cellular decompositions considered by this approach becomes too large, we only take those induced by the direction of long edges, since these directions will have a larger chance of being optimal in reducing the amount of turns taken.

For step 4 we use a travelling-salesman algorithm to find a good route of cells to visit. We consider this in more detail in subsection 3.1.1

In step 5 we cover each individual cell using an extension to a standard back and forth filling pattern. In particular we follow normal back and forth motion, but we extend this pattern to make it possible to actively follow the boundary of each internal obstacle while covering the rest of the cells. This, together with step 2 of the algorithm ensures that the robot has followed all the boundaries each obstacles, making an additional pass through the environment to do so unnecessary. In figure 3 the effect of not making this additional pass can be seen, while the first picture in figure 6 gives an indication of the coverage when including the additional pass. Do note that this addition breaks the robots usual habit of staying within one cell until it is fully cleaned.

Therefore the robot keeps track of each obstacle of which it has already followed its boundary. Then when the robot moves to a location that is immediately adjacent to a boundary of an object that has not been circles around yet, it first goes around the boundary of the entire obstacle to end up back in the same place it started. Then it simply continues its normal back and forth motion within the cell. The fact that this additional motion results in the robot coming back to where it returns, allows this modification to be done regardless of the algorithm used to cover the individual cells.

This additional step allows us to increase the coverage of the robot. The covering algorithm used to fill the cells can also take into account that the boundary of every obstacle will already be covered to decrease the amount of redundant coverage. (An added advantage of this approach is that it also helps robots without the ability to turn on the spot easily, since it can now take advantage of the fact that the outer bounds of the cell are already covered to make a smoother path.) This additional step does not change the ending location of the cell coverage algorithm, allowing it to have no impact on the routes taken between cells.

3.1.1 Optimalization of the cell visiting sequence

In this subsection we will describe how to create an optimal path for traveling between the cells of a (Boustrophedon) cellular decomposition. We do so by taking into account to which location within a cell we want to go to in order to cover that cell.

To calculate which path is optimal a metric for the performance of the robot is needed. For clarity of the argument we will consequently use distance as our metric of performance.

We want to specify the problem as a traveling salesman problem. To do so we need to make a complete graph with a weight on each of the edges. We choose the distance needed to travel between each cell as the weights. This gives a problem since defining a distance between two cells is ambiguous: The problem is than this distance is dependant on where the coverage algorithm starts and stops in a given cell. An approximation can be to take the distance from the centre of one cell to the closest point of another cell (this approximation can be useful when spiral coverage is used for covering each cell, since a spiral pattern can start at an arbitrary edge of the cell and usually stops somewhere near the centre). When using back and forth motions however, we know that the coverage algorithm will always starts and ends at a corner. We can use this information to reduce the problem of specifying distances between cells to distances between points. We will see this will results in a generalized travelling salesman problem, although the vertices of the graph created that way will not represent cells anymore.

We begin by indicating the potential advantage of considering the begin and end points of the cell coverage algorithm in measuring the distances between cells.

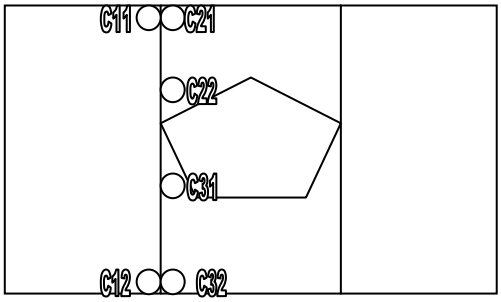

Figure 5. An abstract cellular decomposition.

When we split a cell into two new cells by the Boustrophedon cellular decomposition, we create six new corner points. See Figure 5 for an illustrative example. c11 and c12 are the last two corners of previous cell, while c21 and c22 respectfully c32 and c33 are the first corners of the newly created cells. In this case we know c11 will have minimal distance to c21, and c12 will have minimal distance to c32, while the other distances depend on the environment and can be a lot longer (one can make examples where any of those distances are arbitrarily large, although the rooms size will have to scale with the same factor). Also the distance between c22 and c31 can be a lot smaller than the distance from the centre of cell 3 to some point in cell 1, not to mention that the travelling salesman approximation would also add the distance from the centre of cell 1 to the closest point in cell 2. Therefore when devising a path through the list of cells it can be useful to take into account how the used coverage algorithm, for covering the individual cells, maps the starting location to the ending location within a cell. By doing that, we can create a shorter path for the robot to follow, since we have better information on the distance travelled by travelling from one cell to another which we can then be minimized more effectively.

Suppose we have a single cell from a cellular decomposition and a coverage algorithm that can be applied to that cell. If the algorithm is deterministic and we specify the location of the cell in which we start the coverage algorithm, we can calculate the location in which the algorithm ends (e.g. let the coverage algorithm give the set of location where it will move to (to cover the cell) and take the last location). Therefore if we specify an exact starting location within the cell and a specific coverage algorithm to cover the cell we can get the exact distances between cells. The graph we attain will be a directed complete graph where every vertex represents a cell with a specific start and ending point and the weight on an edge from cell C1 to cell C2 is the distance between the ending point of C1 and the starting point in C2. Of course this seems like just another kind of approximation for the distance between cells, and indeed it can be used that way. However the only limiting factor now is that we chose a specific starting position from the start. If we chose another starting position we would have different weights in our graph. In fact we can expand our graph to allow different starting locations within a cell. A cell will now be a cluster of vertices. Each vertex in that cluster corresponds to that cell and has a different starting location. We now need to find a closed path that includes at least one vertex of each cluster with a minimal length. (We can simply set the distance between points of the same cell at infinity).

This is exactly the description of the generalized travelling salesman problem (GTSP). In some cases we might want to search for a path instead of a cycle, but we can resolve this by changing the weighs of the graph. Once we know the cell we start at we simply change the cost of all edges back to the vertices corresponding to that cell to 0 (all other weights are positive). Once we find a shortest cycle in the GTSP we simply remove the edge with weight 0 to obtain the path we want (we do not care what our endpoint is). (There are many other variations for the TSP including the traveling salesman path problem dealing with an optimal path being given the initial and ending vertices [7,3], which may also be useful depending on the context wherein the robot needs to clean.) The GTSP can be reduced to a travelling salesman problem (TSP) with exactly the same number of the vertices [2]. Therefore the input size to the corresponding TSP we create this way is linear to the number of different starting location we want to minimize over per cell. Since the TSP problem is NP-hard the downside of this linear growth of the problem means that the running time can increase exponentially, although good algorithms are available for solving the TSP problem.

A small modification can be made to the weights of the edges of the GTSP. We want to include different starting locations within each cell as it can decrease the distance we need to travel. So far we have only modeled the distance travelled from cell to cell. Although minimizing this distance is important, we can also minimize the total distance travelled. If we have a coverage algorithm and a cell we can not expect the coverage algorithm to cover exactly the same distance regardless of its starting location within the cell. Do note that computing this distance may have a higher complexity than computing the ending location of an algorithm, as can be the case for back and forth motion.

We can easily account for the actual distance travelled. To do so we can add the total distance travelled by covering a cell C with a particular starting point to all the edges going into the corresponding vertex.

One nice free result of this idea is that we can plan a path taking into account that we may have multiple cell coverage algorithms at our disposal. In our GTSP all the vertices in a cluster were in fact a representation of a way to clean that cell. In our previous discussion we had the same covering algorithm and a different starting location, however any combination of coverage algorithm and starting location can be used.

The downside of this approach is that we the input size of our GTSP grows linearly with the number of (starting point, coverage algorithm) combinations we want to try for each cell. Note however that we are making a global planning, which means that once the global planning is made, the path taken in a single cell may still be optimized locally. We can therefore take a number of representative starting locations when trying to cover the cell with a spiral pattern (basically a very rough discretisation of the boundary). Additionally some cell coverage algorithms only use a small set of starting locations in which case we can actually optimize over all possible starting positions. Back and forth motions as commonly used with the Boustrophedon cellular decomposition are an example of that.

When using simple back and forth motions in a cell created by the Boustrophedon cellular decomposition the usual starting points are the corners of the cell. We therefore only need four vertices per cell in our GTSP in order to plan a path that minimizes over all possible ways to cover all the cells using back and forth motions in a single direction. (Using back and forth motions in a different direction is usually not easily achievable due to the way the cells were constructed). Therefore our improved optimalization problem compared to that used in the original Boustrophedon cellular decomposition in [6] is a GTSP with 4n vertices, where n is the number of vertices in the TSP of the original Boustrophedon cellular decomposition. Since the GTSP can be converted into a TSP with the same amount of vertices, we have increased the problem size by a factor four. Do note that finding a corresponding ending corner for a given starting corner can be done in constant time for this movement pattern, since it is solely determined by the length of the cell in the direction perpendicular to the sweep direction.

3.2 Dealing with unknown environments

Normally a cleaning robot need not have a map of the environment and even if it has one it will not automatically know its location on that map. To handle this, our algorithm as described in section 3.1 needs to be adapted to handle this. First we can note that step 2 still remains trivial except for

finding out when the boundary has been completely followed. While going past the boundary of the environment however, the robot will now collect data about the environment to build a map. In this paper we will not describe how to detect when the robot is back at its initial location on the boundary, which is not a trivial problem. Please see [11] for more information on closing loops.

Ideally the robot can gain knowledge about the entire environment in step 2. In reality this will not always be the case. However the entire map does not need to be known when trying to make a Boustrophedon Cellular Decomposition. To make such a decomposition we assume than the unknown area that is not known to be unreachable (we do not count areas enclosed by walls) is reachable when creating the decomposition. This will make certain boundaries appear to be infinitely thin. However it will not add any large obstacle line that is not to another large line that was already available in the obstacle boundaries that are correctly mapped. Therefore this will not increase the amount of directions over which we need to optimize in step 3 (Although due to a lack of data each optimalization step will be further away from optimal then in completely known environments).

In step 5 covering every completely known cell can be done similarly as before, although now needing to correct for sensor and localization errors. Most importantly though the robot will still collect additional information about the environment, which it can use to gain information about other cells. Our extension of the back and forth motion pattern to include going around obstacles is ideal for this. For unknown environments we gain additional information by moving along the boundary of an obstacle. If our sensor range is large enough, by following this procedure, we suspect we can always find an uncovered cell which is fully known. Therefore covering a single cell comes down to being able to cover a known cell and moving along the boundary of an obstacle up to the point where boundary following started. Once unknown area is discovered however, we might need to split the cell in which it belonged up, such that every cell is coverable by back and forth motions. This can be done by locally recreating the Boustrophedon Cellular Decomposition for the cell with the unknown area.

In an unknown environment step 1 of the algorithm is less straightforward, because we can not use classical path planning. One way to handle this problem is to go to the nearest boundary of the environment, going around its boundary and then deducing whether it was the outer boundary or not based on the path taken. If it was not the outer boundary this procedure can be continued any number of times, constantly taken a nearest boundary edges that has not been traversed yet, until we managed to go around the outer boundary. This is may not seem very efficient, but the obstacles whose boundaries we already covered can already be marked as being done, which decreases the amount of work done in step 5. Effectively the robot simply tries doing step 2 until it succeeds, remembering the additional obstacles it went around, to prevent it from moving around them again in step 5.

Step 4 of the algorithm also becomes more complicated in unknown environments. Since not all the cells are completely known in advance it is hard to optimize the sequence of cells to visit. Globally finding an optimal sequence as in the known environment can not be done on the same precision. Specifically our own approach as in section 3.1.1 can not be directly applied, because the end locations are no longer deterministic given an initial location and a description of the cell (because of uncertainty in localization as well as in the map). Expected values for the end location can be used to remedy this problem. However to get a decent path it might be useful to heuristically determine a path during the cleaning instead of calculating one in advance. Doing so will prevent lack of knowledge at the start from giving an inefficient visiting sequence of the cells. (Once dynamic environments are considered this will also be useful to avoid doing a lot of unnecessary computations). This will need to take into account the trade-off of going to the nearest corner of an uncovered cell (which a greedy approach would do) and not getting large movement costs in the long run, by leaving multiple isolated cells that still need to be cleaned at large distances from each other.

3.3 Dealing with dynamical changes in the environment

In this section we distinguish between moving obstacles like humans and pets from static objects than have been moved, since they should be handled differently. We first describe how the second case is handled.

3.3.1 A static obstacle that has been moved.

Here wet consider the event that an object suddenly completely appears/disappears. Every movement can after all be seen as a removing an object and immediately afterwards adding the object at another location (usually rotated a bit). Additionally this situations automatically occurs when someone picks up an object or drops it. Handling these two cases is therefore sufficient for handling static objects being moved to different locations.

Our approach resolves around locally creating a new cellular decomposition. When an object is removed from the environment, a simple way to deal with this change would be to make a decomposition of the additional area that now needs to be cleaned and adding the cells created that way to the current collection of cells, after which the path between cells needs to be updated. However if an object is removed from the environment some of its surrounding cells (and the area it was occupying itself) may possibly be merged to bigger cells, which increases efficiency of the created path.

To do so we take all the neighboring cells of the object together with the area it occupied and create a new Boustrophedon Cellular decomposition for them on the new environment where the obstacle was removed. However we treat the already cleaned cells as obstacles, to avoid merging cleaned cells and unclean cells together, because we want our cells to be either cleaned or not cleaned. There may be one cell among the cells under consideration that is partially cleaned, because we were busy cleaning that cell when the obstacle suddenly disappeared. In that case we split that cell up into two cells. One cell with its cleaned area, which we will consider as an obstacle for taking the new decomposition and one cell with the area that still needed to be cleaned. For every merged cell we remove the old cells that were a part of it and mark the new cell as unclean.

The area where the obstacle was could have been partially cleaned already if it did not stand there the whole time and we managed to clean a part of the area where it stood already. The robot needs to keep track of this kind of information, to avoid repetitive cleaning in situations where repositioning of obstacles happens a lot.

We basically treat cleaned area as obstacles for this recreating of the cells to prevent the robot for covering area's in already covered. In the case of small cleaned areas that are in the area that was formally covered by the obstacle it can prove useful to

consider them unclean in order to improve the efficiency of the path (as long as the area is small the redundant coverage created this way will be small too, while we avoid inefficiency of smaller cells by allowing them to be merged). Additionally we do not want to store unnecessary large amounts of data, giving us another reason to merge small cleaned cells created this way with the unclean cells.

Now that the decomposition of the environment has been changed, the sequence of cells to visit changes as well. Determining a new sequence can be done in the same way as with the static case, but now only taking the cells that have not been cleaned yet into account.

The case of a suddenly appearing obstacle is handled similarly. If an object is suddenly added to the environment we need to split a number of cells and we might be able to merge some back together (the cells in which the obstacle appears will generally stop being simple areas that can be covered by simple back and forth motions in the direction of the sweep line). To do so we take all the cells intersecting with the new obstacle and create a new Boustrophedon Cellular decomposition for them on the new environment with the added obstacle for these particular cells. However we treat the already cleaned cells as obstacles, since we do not need a decomposition of an area we do not need to cover anymore. Note that as a special case we do not need to do any work if a new obstacle appears in an area we have already covered. There may be one cell among the cells under consideration that is partially cleaned, because we were busy cleaning that cell when the obstacle suddenly disappeared. In that case we split that cell up into two cells (again). One cell with its cleaned area, which we will consider as an obstacle for taking the new decomposition, and one cell with the area that still needed to be cleaned. Now we simply remove all the old uncovered cells for which we created a new decomposition and add the new cells.

3.3.2 Moving obstacles

Moving obstacles can in theory be handled like static objects being moved small distances at a time over a long time period. However this is very inefficient due to a constant need for recreating a part of the cellular decomposition. Additionally doing so will not allow the robot to avoid collisions with the moving obstacles, because it will assume at each time that the obstacle is not moving. It can therefore go very close to that obstacle. If that obstacle was a person this kind of behavior could make him/her trip, which is unwanted. Finally as long as the obstacle moves it will only be a matter of time until it leaves the cell it currently occupies, meaning that there may not be any need to change the cellular decomposition of the cells it occupies, since the original decomposition may be applicable again after a small amount of time passes. Due to these considerations we see a need to handle moving obstacles differently.

Moving obstacles will not be used in creating a cellular decomposition of the environment. Once a moving objects stops moving for about five seconds, it will then be treated as a static obstacle and will be treated as an obstacle that suddenly appeared (as in section 3.3.1). If a static obstacle suddenly starts moving it will be treated as if the static was suddenly removed. We do note though that in an unknown environment classifying an obstacle as moving based on raw sensor input is not trivial. It is however beyond the scope of this paper.

Moving obstacles are handled by making the cleaning robot avoid going close to moving obstacles, to prevent collisions. Only a simple approach for doing this is presented here. Once the robot is covering a cell and an obstacles is moving close to a location it needs to go to, it will wait until the moving obstacle goes out of that area. If that does not happen. after a certain amount of time it will be treated as static allowing the robot to continue. It will then be allowed to go up close to the previously moving object. If the obstacle goes towards the robot, the robot will try to evade it. To do so it will try to move along a line orthogonal to the direction of the moving object, while choosing the direction along that line such that the distance between the robot and that obstacle is increased. If that direction is blocked (by either static obstacles or the presence of other moving obstacles) or if there is not enough time to evade the moving obstacle that way, an weighted average of that direction and the direction in which the obstacle is going will be considered. Evading a moving obstacle will be done in small steps. If at any time all the considered directions are blocked (usually at a corner, but with multiple moving obstacles other scenarios can occur as well) the robot will wait hoping for the better.

4. VERIFICATION

To verify the correctness of the algorithm a part of it has been simulated. Although the direction of the back and forth motions has impact on the efficiency, our implementation always uses a vertical sweep line as in figure 1. Square environments with proportions of 20r by 20r, with r being the radius of the robot, were used.

There are two common ways to model the environment in which the algorithm should work. One way is to approximate the environment with a grid. Optimal coverage then comes down to visiting all grid locations. In this model the robot itself usually occupies a single grid space. The second approach models all the obstacles by specifying their boundaries. Although more complicated, the second method prevents round off errors and robots using this model can therefore have a higher coverage ratio than robots using a grid. In our simulation we therefore used the second approach to model the environment.

A simple algorithm to decide the path between cells was implemented. It makes use of the already computed Boustrophedon cellular decomposition to calculate a path and can only plan a route between a starting location in one cell and a target location in one of its adjacent cells.

The list of cells to be created in step 4 of the algorithms is not being made in advance, but is expanded by one new cell every time the robot finishes cleaning its current cell. To determine the next cell to visit it uses Dijkstra´s algorithm to take a cell that is closest to the current cell as the next cell. The advantage of using Dijkstra is that, by storing the usual predecessor data for each cell, we can create a list of cells the robot needs to traverse in order to get to the target cell. This list can thus be used in combination with the path planning algorithm that can be used to calculate a path between adjacent cells, to get a path between two arbitrary cells (assuming the cells are reachable).

For adjusting the Boustrophedon cellular decomposition an approximations of the algorithm in section 3.3.1 was used. We recomputed the whole decomposition instead of a small piece of it, to avoid the implementation detail of only recreating the local cells. We also explicitly matched the old cells and used this matching to explicitly split and merge cells as required. However this did not change the path the robot followed in the examples we used.

We have simulated the event of an object instantly appearing under several situations in order to verify that the algorithm did not make the robot stop until if could not achieve maximal coverage:

- The object appears in an already cleaned cell
- The object appears in an cell that still needs to be cleaned.
- The object appears in the cell the robot is cleaning at the time
- A part of the object appears in an already covered cell and an other part appears in a cell that still needs to be cleaned.

In each of these situations the environment was known. In all cases the already cleaned regions of the environment were not covered using back and forth motions again. In all cases the algorithm achieved maximal coverage. Additionally the approach did not introduce additional path overlap in requiring back and forth motions to be planned needlessly.

Within the simulation the cleaning path made by the robot was recorded. This path was used to calculate the total distance travelled, the number of turns taken, the sum of the angles the robot has turned and the total area that has been covered. This information was used to compare the efficiency of our approach compared to random path planning.

In figure 6 the path taken in one of our testing environments is shown for our proposed method and for random path planning. We ensured that the distance traversed by the random path planning algorithm equals that of our algorithm. The obstacle to the bottom left appeared during simulation (which is why the robot could partially clean the area occupied by the obstacle). In order to compare our proposed algorithm to the random path planning algorithm we run 100 simulations of the random path planning approach where we left every parameter the same, except for the random numbers the path planner used to determine the next direction in which the robot would move.

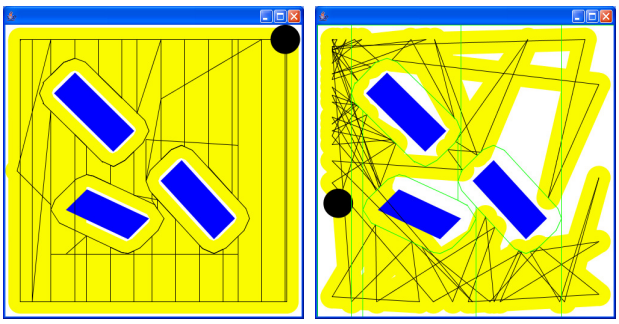

Figure 6. A path planning example in a room that has dynamical changes.

(left: our algorithm, right: random path planning).

In table 1 the results for this particular environment can be seen. For the random path planning approach 95% intervals are given along with the average value. We can conclude that random path planning needs to turn less with the same distance travelled than when using our proposed algorithm for this environment. Its coverage is however considerably lower. We also has environments where our proposed algorithm on average needed to turn less than random path planning. These environments has less obstacles, and the boundaries of their obstacles were better aligned with the boundary of the room.

Table 1. Simulation results. LB and RB give the left and right bounds for the corresponding confidence interval.

	Cellular decomposition	Random		
	average	average	LB	RB
distance	21716	21718	21718	21718
turns made	144	73,1	71,4	74,8
radians turned	206	162,0	158,1	165,9
area covered	865946	648061	637973	658148

We simulated the random approach with different distances and as the distance travelled by the robot increases, random path planning increases its coverage as can be seen in table 2 (which it should, since the literature states that random path planning has maximal coverage as the cleaning time goes to infinity [12]). However as table 2 also indicates, at a certain point our exact cellular decomposition outperforms random path planning in every aspect. This is due to the exponential shape of the area a random path planning algorithm covers [12], while all the other performance aspects increase linearly in the time spend cleaning.

Table 2. Simulation results with the random path planning travelling double the distance.

	Random with a distance of 43437		
	average	LB	RB
turns made	144,8	142,2	147,4
radians turned	321,0	314,9	327,1
area covered	788660	781740	795580

We also simulated other environments. In some environments we got similar results and in some other environments our algorithm outperformed the random path planning approach in every aspect.

5. CONCLUSIONS AND FUTURE WORK

In this paper a small overview of some coverage path planning algorithms was given in section 2 along with their strengths and weaknesses. Most coverage algorithms use a cellular decomposition to ensure complete coverage. In this work the Boustrophedon Cellular Decomposition was expanded to handle dynamic, unknown environments to create a coverage path planning algorithm suitable for cleaning robots. We first showed how for a static known room an optimal sequence in which to cover every cell of an environment can be computed, by formulating the corresponding optimalization problem as a travelling salesman problem. Additionally we specified how the algorithm can be adapted to follow the contour of every obstacle without needing to do it as a second pass and without requiring a specific way to clean the cells of the decomposition. A couple of the difficulties in handling an unknown and dynamical environment were described. For unknown dynamic environments a near optimal sequence of cells to visit can not easily be calculated at the start of the cleaning routine and an online algorithm might therefore have better results. In this paper a simple algorithm based on Dijkstra´s algorithm was used. In dynamical environments the Boustrophedon Cellular Decomposition can still be used, but it needs to be updated whenever the robots representation of the environment changes. This paper describes how the decomposition can be updated in an efficient way. Our algorithms attained complete coverage in a number of environments, including environments where obstacles were added during the simulation. From our simulations we conclude that for some environments random path planning performed better when the required coverage is

70% or less. When needing coverage above 85% our algorithm generally performed better regardless of the environment. In our simulation we used a path planning algorithm that regularly takes detours, while we avoided large rooms (random algorithms tend to be less efficient at cleaning large rooms). These results therefore confirm that our algorithms performs better than random path planning when near optimal coverage is desired.

Due to time constraints a significant portion of the specified algorithm has not been explicitly verified using simulations. In future work the specified algorithm still needs to be tested in more general situations including unknown environments and environments with more types of dynamic behavior.

As was mentioned in section 3.3.1, an obstacle that moved only in a small magnitude (a small translation, a small rotation or both) can simply be seen as removing the object and adding a slightly different object. In unknown environments this situation can occur due to our SLAM algorithm making corrections to the internal map based on new information about the environment. Therefore this situation can occur often enough to make optimalization of this special case useful to decrease the amount of computations the robot needs to do.

Furthermore in this paper a couple of suboptimal algorithms were used for things like the path planning between cells, and the algorithm was only compared with a random path planning approach. To gain more insight in the usefulness of our approach an optimized implementation should be compared with additional coverage path planning algorithms.

6. ACKNOWLEDGEMENTS

I would like to thank my supervisors, Anton Stoorvogel and Mannes Poel, for their supervision and feedback during this research project.

7. REFERENCES

[1] Acar E. U., Choset H., Zhang Y., Schervish M. “Path Planning for Robotic Demining: Robust Sensor-Based Coverage of Unstructured Environments and Probabilistic Methods” The International Journal of Robotics Research July 2003 vol. 22 no. 7-8 441-466

[2] Behzad A, Modarres M. "A New Efficient Transformation of Generalized Traveling Salesman Problem into Traveling Salesman Problem" Information Sciences, Volume 102 Issue 1-4, Nov. 1997

[3] Chekuri C., Pál M."An O(log n) Approximation Ratio for the Asymmetric Traveling Salesman Path Problem", THEORY OF COMPUTING, Volume 3 (2007) Article 10 pp. 197-209

[4] Chi J., Yang G., Yang J.. "The on-line coverage algorithm and localization technique of the intelligent cleaning robot” Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference, 6-8 April 2008, page(s): 943 - 948

[5] Choset H. "Coverage for robotics – A survey of recent results", Annals of Mathematics and Artificial Intelligence 31: 113–126, 2001.

[6] Choset H., Pignon P. "Coverage Path Planning: The Boustrophedon Cellular Decomposition", International Conference on Field and Service Robotics 1997

[7] Friggstad Z., Salavatipour M.R., Svitkina Z., "Asymmetric traveling salesman path and directed latency problems", Proceeding SODA '10 Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms

[8] Gonzalez, E. Alarcon, M. Aristizabal, P. Parra, C. “BSA: a coverage algorithm“, Intelligent Robots and Systems, 2003. IEEE/RSJ International Conference, 27-31 Oct. 2003, page(s): 1679 - 1684 vol.2

[9] Gonzalez, E. Alvarez, O. Diaz, Y. Parra, C. Bustacara, C. "BSA: A Complete Coverage Algorithm", Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference, 18-22 April 2005, page(s): 2040 – 2044

[10] Huang, W.H. "Optimal line-sweep-based decompositions for coverage algorithms" Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference, Issue Date: 2001 page(s): 27 - 32 vol.1

[11] Kristopher R. Beevers "mapping with limited sensing", Thesis (M. Eng.)--Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2008.

[12] Palleja T., Tresanchez M., Teixido M., Palacin J.. "Modeling floor-cleaning coverage performances of some domestic mobile robots in a reduced scenario", Robotics and Autonomous Systems , Volume 58 Issue 1, January, 2010 ,pp. 37-45

[13] Prassler E., Ritter A., Schaffer C., Fiorini P.."A Short History of Cleaning Robots", Autonomous Robots, Volume 9 Issue 3, December 2000,pp. 211 – 226

[14] Yang S.X., “A neural network approach to complete coverage path planning”, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions, Feb. 2004, Volume: 34 Issue:1, page(s): 718 – 724

[15] Ja-Young Sung1, Rebecca E. Grinter1, Henrik I. Christensen1, Lan Guo. "Housewives or Technophiles?: Understanding Domestic Robot Owners" HRI’08, March 12–15, 2008, Amsterdam