# Automated Vacuum Cleaner Study

**Rudra Sharma** and **Joshua O'Dell**
Colorado State University
Fort Collins, CO 80523

## Introduction

The purpose of this project is to simulate an automated vacuum cleaner (AUC). We represent a room as a 2D plane which has clean and dirty points/tiles. The goal of the vacuum cleaner is to clear all the dirty points and navigate/find a path around the the room.

In the project, we implement a variety of different vacuums and study its impact on performance. There have been several papers written on the study of automated vacuum cleaners. IRobots Roomba[1] is an example of such a robot, we will study this implementation when developing our own algorithms.

Yap [2] explores a variety of grid based path finding algorithms, they identify an octet selection grid as an option of choosing the next point for the robot to move to as shown in figure 1. While the direct application of this paper will apply to room cleaning vacuum, these path-finding algorithms can be applied to robots searching for landmines, lawnmowers, and other agricultural tasks [3].

The implementation for this this project will be derived from a rudimentary implementation of a roomba robot [4]. This implementation [4] chooses its next tile based on a random selection from a 360° scale. We have instead opted to use the octet based selection as shown on figure 1 to choose the next tile.

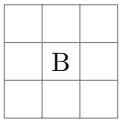


Figure 1: octet choice grid: the bot,B, has eight adjacent positions it can move to

## Experiment Implementation

The project implemented six distinct vacuum robots. Each Robot has a different set of percepts, and the algorithm is adjusted to deal with it. The algorithms in each bot is a greedy algorithm with a different heuristic for calculating the cost based on the percepts it has.

The following is a list of the capabilities of each Vacuum robot.

1. **Preloaded Map:** Before the cleaning begins a map of the world is loaded into each the vacuum robot.
2. **Store explored nodes:** As the vacuum robot moves through the world it is allowed to store without limit, the tiles where it has been, and the status of that tile
3. **Proximity Sensor:** the vacuum robot has the ability to know when it is close to a wall.

### Random Bot

| | | | |
|---|---|---|---|
| | | Observable | Unobservable |
| Preloaded map | No | Deterministic | Deterministic |
| Store explored | No | Episodic | Sequential |
| Proximity sensor | No | Static | Static |
| | | Discrete | Discrete |
| | | Agents | Single |

The Random robot was the first Robot we implemented for our study. This robot has no form of storage or sensors to guide it in cleaning a room. When the robot is initialized, it will clean the location it is placed on and then it will randomly select a next point based on what is adjacent to it, thereby giving it eight options. See figure 1 for display of the octet grid selection for the random robot.

### Store Map Bot

| | | | |
|---|---|---|---|
| | | Observable | Fully |
| Preloaded Map | Yes | Deterministic | Deterministic |
| Store explored | No | Episodic | Sequential |
| Proximity sensor | No | Static | Static |
| | | Discrete | Discrete |
| | | Agents | Single |

The preloaded map was a Robot which was given the the exact boundaries of the room prior to start. The objective of this robot was to use its knowledge of the boundaries of the room to avoid attempting to go outside the boundaries of the room. This robot was an extension to the random robot so the path that it choose next was also a random selection from the

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

octet grid, however it knows to avoid the points which represent tiles outside the boundaries of the room.

### Store Map with Direction Bot

| | | | |
|---|---|---|---|
| | | Observable | Fully |
| Preloaded Map | Yes | Deterministic | Deterministic |
| Store explored | No | Episodic | Sequential |
| Proximity | No | Static | Static |
| sensor | | Discrete | Discrete |
| | | Agents | Single |

The store map with direction was an extension to the store map. However, this map no longer had a random element which decided its next point in the room. Instead it uses a strategy of maintaining a direction until it reaches the boundary of a room. After it reaches a boundary it will side to the adjacent slide and do a 180° turn and travel in the the opposite direction it came from until it hits the opposite boundary of the room. It will keep repeating this process until it completes the entire room.

### Proximity Bot

| | | | |
|---|---|---|---|
| | | Observable | Partially |
| Preloaded Map | No | Deterministic | Deterministic |
| Store Explored | No | Episodic | Sequential |
| Proximity | Yes | Static | Static |
| sensor | | Discrete | Discrete |
| | | Agents | Single |

This program was implemented by mimicking the 'Roomba"s algorithm [1]. The algorithm creates a spiral from the starting position outward. Once the proximity sensor detects a wall the algorithm changes. The new algorithm will choose a random direction, and continue straight along that direction until the proximity sensor detects another wall, at which point another random direction is chosen.

The largest hurdle with this program was finding the best equation to calculate a spiral. At each step in the simulation a new direction was calculated using the following equation. $22.5 - (e^x * 45)$ where time x was decremented by 0.01 during each step. The program appeared to be efficient, however when run, we saw that it had difficulty achieving 10% coverage in a timely manner. We attribute this to some tweaking that is necessary on the spiral formula. We believe it is going over the same places too much, however given our grid approach any more aggressive spiraling would result in a missed grid tiles in the center.

### Stored Explored Bot

| | | | |
|---|---|---|---|
| | | Observable | Unobservable |
| Preloaded Map | No | Deterministic | Deterministic |
| Store explored | Yes | Episodic | Sequential |
| Proximity | No | Static | Static |
| sensor | | Discrete | Discrete |
| | | Agents | Single |

This program is allowed to maintain a list of the grid spaces where it has visited. It is allowed to know where each of these grid places are in relation to its current location. The algorithm used in this program will attempt keep the vacuum close to the grid items that were previously visited. To do this the following steps are taken.

1. Get a list of all the possible steps we can take.
2. For each step determine a ranking. This is calculated by summing the distance to each of the existing items.
3. Choose the step with the highest rank.

If a spot is chosen that is outside of the grid, the programs time complexity ranking is incremented, however the robot will not be able to move there. We also try to avoid revisiting gird points where we have already been visited once. To do this if a possible location turns up in our visited locations, then we increase the rank. Essentially moving the rank high enough that other options, even if they move away from the existing cluster, will be chosen. Thus, the rank equation is as follows.

$$Rank = \begin{cases} visited = False : \sum distance(visited) \\ visited = True : sizeOf(Visited) \end{cases}$$

### Store Explored + Proximity Bot

| | | | |
|---|---|---|---|
| | | Observable | Partially |
| Preloaded Map | No | Deterministic | Deterministic |
| Store explored | Yes | Episodic | Sequential |
| Proximity | Yes | Static | Static |
| sensor | | Discrete | Discrete |
| | | Agents | Single |

This program is an extension of the 'store explored' program. An observation about made about the store explored was that depending on where it started, we could get into a situation where we had finished the middle of the room, and one side, but needed to complete the other side. To do this we have to cross the already cleaned region, which our algorithm did not handle elegantly.

Adding a proximity sensor the the vacuum robot allowed us to do the following. When the robot first starts cleaning it chooses a single direction. Once it reaches a wall it then begins the existing algorithm from the 'store explored' robot. Then this robot can start cleaning on one side of the room, and not have to back track. The result was improved cleaning times.

## Results

### Experiment 1

We began our data gathering by running each robot and asking it to clean a certain percentage of the room. The Robot was given 1,000,000 steps to complete the task. Table:**??** describes our results of this initial experiment.

The results of this experiment show us that at 1,000,000 steps most of the robots can succeed even if the requirement is to clean 100% of the room. However, the above table hides a key element 'how long did it actually take'. Table:**??** shows in more detail how many iterations each robot actually took to complete

| | 10% | 50% | 80% | 100% |
|---|---|---|---|---|
| Random | 96% | 77% | 40% | 12% |
| Store Map | 100% | 100% | 100% | 100% |
| Store Map with direction | 100% | 100% | 100% | 100% |
| Proximity | 40% | 34% | 27% | 25% |
| Store Explored | 100% | 100% | 100% | 100% |
| Store Explored with proximity | 100% | 100% | 100% | 100% |

Table 1: Percent success (depth = 1,000,000)

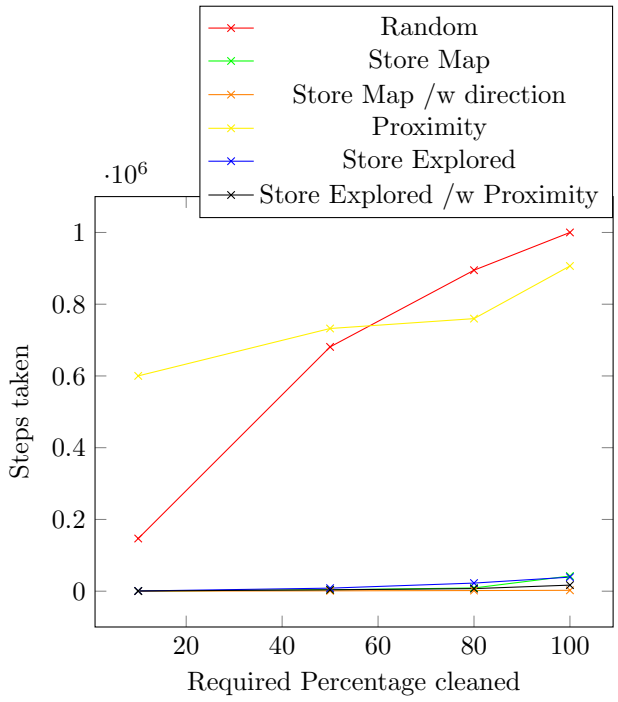


Figure 2: Steps taken by each Robot to complete the required percentage (depth = 1,000,000)

| | 10% | 50% | 80% | 100% |
|---|---|---|---|---|
| Random | 146783 | 680976 | 894746 | 1000000 |
| Store Map | 373 | 3895 | 9006 | 42284 |
| Store Map with direction | 160 | 947 | 1756 | 2408 |
| Proximity | 600159 | 732261 | 759790 | 906491 |
| Store Explored | 535 | 8491 | 22586 | 39161 |
| Store Explored with proximity | 539 | 3857 | 7163 | 16898 |

Table 2: Actual steps taken to complete task (depth = 1,000,000)

the task. The numbers provided are an average of 100 runs, this average includes the failures and successes. If a robot was unable to complete the task then its step count would reach 1,000,000 and halt, indicating failure to achieve coverage, and contribute to bringing the average processing steps higher. Figure:**??** shows this in a graphical format. We can see that the Random and Proximity robots initially took the longest to complete their tasks, and continued to do so as the required percentage increased.

### Experiment 2

However 1,000,000 iterations of the program does not allow the robot to complete in an acceptable amount of time. If we assume that it takes one second for a robot to clean a square and move on, then each robot would take close to $1000000/60/60/24 = 11.5 days$ to complete it's task.

Our second experiment took this into account and lowered the maximum steps to 10,000. Table:**??** shows the outcome of this, and Figure:**??** shows this graphically.

| | 10% | 50% | 80% | 100% |
|---|---|---|---|---|
| Random | 96% | 77% | 40% | 12% |
| Store Map | 100% | 100% | 100% | 100% |
| Store Map with direction | 100% | 100% | 100% | 100% |
| Proximity | 40% | 34% | 27% | 25% |
| Store Explored | 100% | 100% | 100% | 100% |
| Store Explored with proximity | 100% | 100% | 100% | 100% |

Table 3: Percent success (depth = 1,000,000)

In this experiment we can see more clearly that most of the robots can complete the task when only a small percentage of the room is required. However, when the coverage percentage increases most robots have fewer successful runs. We see that the 'Random' and 'Proximity' Robots do not do a very good job once they are required to clean more then half of the room. The 'Stored Map w/ Direction' always completes the task, and the 'Store Explored w/ Proximity' does a great job until it

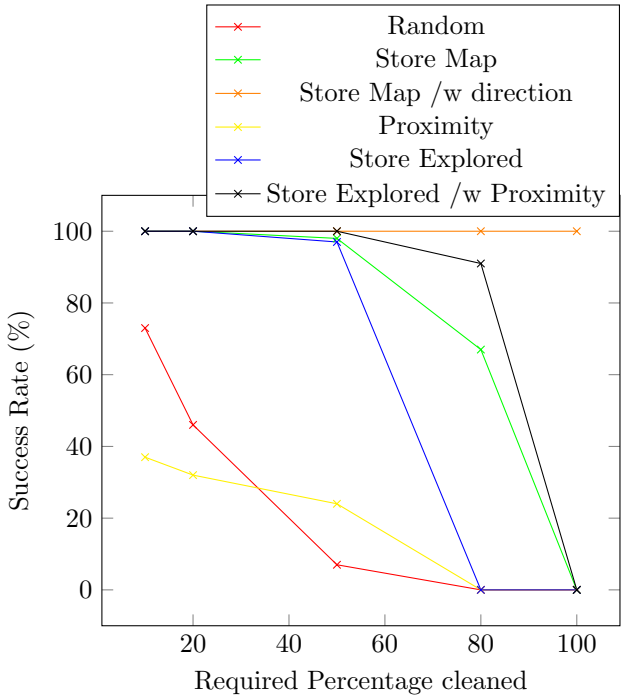


Figure 3: Percentage of successful runs (depth = 10,000)

| | 10% | 50% | 80% | 100% |
|---|---|---|---|---|
| Random | 3570 | 9752 | 10000 | 10000 |
| Store Map | 375 | 3941 | 7905 | 10000 |
| Store Map with direction | 160 | 588 | 1707 | 2403 |
| Proximity | 6431 | 9476 | 10000 | 10000 |
| Store Explored | 520 | 8539 | 10000 | 10000 |
| Store Explored with proximity | 533 | 3864 | 6924 | 10000 |

Table 4: Actual steps taken to complete task (depth = 10,000)

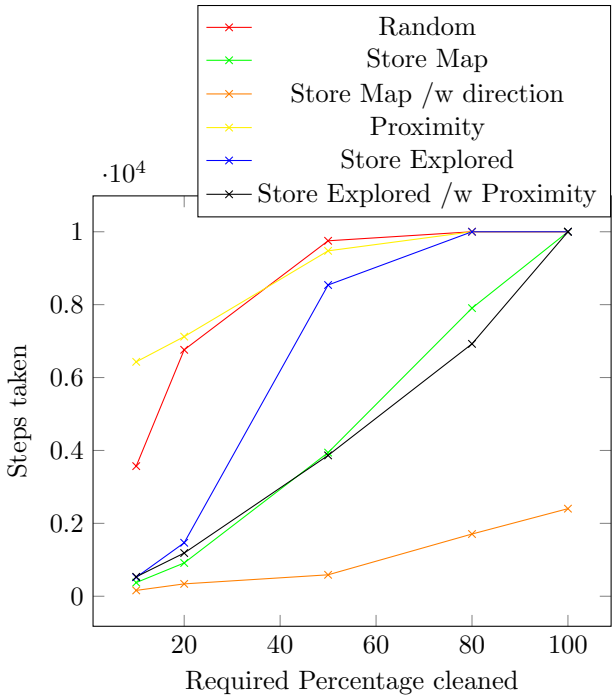


Figure 4: Steps taken by each Robot to complete the required percentage (depth = 10,000)

| | % cleaned |
|---|---|
| Random | 24 |
| Store Map | 82 |
| Store Map with direction | 100 |
| Proximity | 17 |
| Store Explored | 54 |
| Store Explored with proximity | 94 |

Table 5: Actual percentage cleaned (depth = 10,000)

is required to clean the entire room.

### Experiment 3

Running a slightly different experiment we tested to see how close some of these robots came to succeeding when asked to clean the whole room. Table:**??** shows the results of this experiment. The interesting thing to note here is that while 'Store Map' and 'Store Explored /w Proximity' did not complete 100% of the room they got very close at 82% and 94% respectively.

## Conclusion

One of the questions we had when we started this project was 'Can a crippled robot perform as well as an Omnipotent one'? The answer we found out was absolutely not. However given some clever algorithms and relaxing the requirements of 100% clean allows for

these robots to do 'OK'. When the robot is given the map of the world, and knows where it is, then it is easy to come up with a very efficient algorithm to clean every part of the room. However when we removed this ability the results got quite a bit worse. That being said the 'Store Explored /w Proximity' did very well on its own. However, the realization of this robot may not be as realistic as we might hope. Being able to store all the places where the robot has been could be difficult to implement outside of a simulation.

The commercial product most known in this area is the 'Roomba.' The Roomba's algorithm[1] is similar to the 'Proximity' one that we implemented. We found that this algorithm is pretty inefficient, However we were hard pressed to find a better one given the limited percepts. I also believe that this algorithm performs better when measured in a non grid environment. Given the spiral nature of the algorithm it was difficult to map that onto a grid without missing grid positions, or going over them more then once. This robot has no knowledge of where it is or where it has been, it simply switches between two different algorithms based on whether it has hit a wall yet.

There is still a lot of room in this field for better algorithms. Since we have found that knowledge of the room makes a huge difference in finding an efficient path, perhaps a learning robot could be created. One that uses the percepts it has to create a statistical probability of where it is in the room based on the given percepts. We will leave this investigation to future projects.