Scheduling and Motion Planning of iRobot Roomba

Jade Cheng
yucheng@hawaii.edu

Abstract

This paper is concerned with the developing of the next model of Roomba. This paper presents a new feature that allows the robot to resume its work after intervals of charging. This paper also models the robot's working behaviors under different battery storage conditions. We simplify and approximate ancillary parameters while focusing on features associated with the self-navigation system. The key components are expressed as a state machine. We present the entry actions, exit actions, transition actions, and the operation solutions for each state. We also provide possible solutions and algorithms for each state. We do not, however, provide an implementation of the state machine, or test plan with the model.

1. Introduction

This paper is concerned with the developing of new features for the next model of Roomba. This is the investigation of the R&D project for the iRobot. The specific novel feature that we will present in this paper targets the resuming of tasks after battery recharging.

How to teach the robot to resume its work from its last run is a feature that interests customers. As posted on the iRobot forums, customers have asked for related features with comments such as "It would be nice if Roomba remembers that it only cleaned the first several rooms but not all of them in the current cleaning cycle. The next time it runs, it knows that the first several rooms were already cleaned and immediately flag itself internally that the room it was currently in was done and it would start turning on and seeking the Lighthouse tractor beam to traverse to the next room." [1] If we design a new generation of the iRobot that is able to solve this problem, it would greatly benefit the company and the customers.

Hereon we present a description using the concept of a state machine to analyze the challenge presented above as a problem formulation. We make the assumption that a Decision Researcher with a strong mathematical background is the target reader.

This paper will not, however, provide an implementation along with the model and problem formulation. This paper will not provide a test framework, which would be necessary to ensure the model is accurate in more than just theory. Instead, this paper will provide simplified examples, figures, and straightforward state machine expressions that appeal to one's common sense and intuition.

1.1. iRobot Functionality

1.1.1 Sensor system

According to iRobot, the robotic brain of the Roomba can adapt to new input up to 67 times per second. The first thing Roomba does when users press "Clean" is calculate the room size. The iRobot sends out an infrared signal and checks how long it takes to bounce back to the infrared receiver located on its bumper. Once it establishes the size of the room, it knows how long it should spend cleaning.

While Roomba is cleaning, it avoids steps or any other kind of drop-off using four infrared sensors on the front underside of the unit. These cliff sensors constantly send out infrared signals, and Roomba expects them to immediately bounce back. If it approaches a cliff, the signals are suddenly lost, and this is how Roomba knows to reverse its direction. When Roomba knocks into something, its bumper retracts, activating mechanical object sensors that tell Roomba it has encountered an obstacle. It then performs and repeats the sequential actions of backing up, rotating and moving forward until it finds a clear path.

Another infrared sensor, which we call the wall sensor, is located on the right side of the bumper

and lets Roomba follow very closely along walls and around objects, like furniture, without touching them. This means it can clean in close proximity to these obstacles without bumping into them.

The robot determines its own cleaning path using what iRobot calls a pre-set algorithm that achieves complete floor coverage. The Roomba starts cleaning in an outward-moving spiral and then heads for the perimeter of the room. Once it hits an obstacle, it believes it has reached the perimeter of the room. It then cleans along the "perimeter" until it hits another obstacle, at which point it cleans around it, finds a clear path, and proceeds to traverse the room between objects like walls and furniture until the allotted cleaning time is up. The idea is that if it cleans for a certain amount of time, it should cover the entire floor. But whether it actually achieves complete floor coverage is extremely hit-or-miss [2].

1.1.2. Lighthouse Navigation

The Roomba's lighthouse accessories are especially useful, and they are considered in this paper as an essential tool to conduct the new feature. The lighthouse can act as virtual walls to confine the Roomba to a particular room, or they can serve as guideposts to allow the robot to clean one room, travel into the next room to continue cleaning, and then navigate back to the docking station for recharging. There are, therefore, three different services that the navigation system provides [2]. While the lighthouse is actively providing one of these three services, we say it is operating in one of three mutually exclusive modes.

1.1.2.1. Virtual Wall Mode.

In virtual wall mode, the lighthouse turns on automatically, and the robot will never cross its fence beam, which acts like an invisible barrier [2].

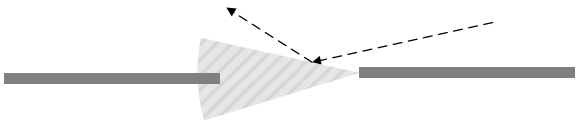

Figure 1. Virtual wall mode

1.1.2.2. Lighthouse Traversing Mode.

In lighthouse traversing mode, the lighthouse acts like a virtual door, which opens when the robot asks it to. Most of the time, the lighthouse has its fence beam turned on, which prevents the robot from going past the lighthouse. Once the robot has finished cleaning the current room, it "asks the lighthouse to open the door," and the robot traverses the lighthouse. Traversing is demonstrated in the following figure [2].

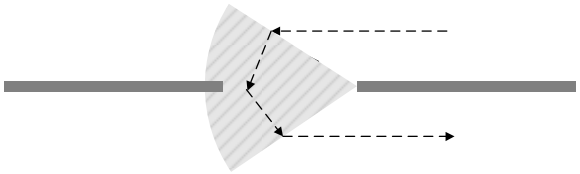

Figure 2. Traversing mode

1.1.2.3. Docking Navigation Mode

When lighthouses are in use, the robot will ignore signals from any and all home bases—the docking station unless the robot thinks that it is in the same room where the dock is located. The robot decides what room it is in based on how many lighthouses it has traversed [2].

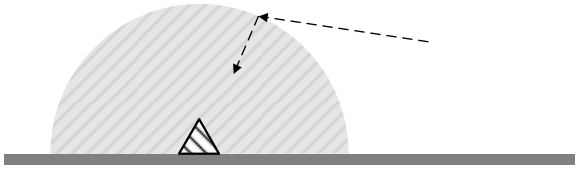

Figure 3. Docking navigation mode

1.1.3. Self-charging

Self-charging is a built-in feature for the relatively newly generations of the iRobot. The machine comes with a docking station that plugs in an AC outlet. The docking station sends out signals that the robot receives and uses to determine the position of the docking station. The communication between the robot and the docking station turns on only when they are in the same room. If there is any other light house between them, the other lighthouse would be turned on before the docking station.

For the moment, we will assume that the robot is able to see the docking beams of the home base, but there is a virtual wall or other impassible obstacle between the robot and the home base. If the robot attempts to follow the docking beams, it will follow

them until it reaches the impassible object and then abort its approach and try again. Thinking it has found the home base, the robot could end up getting stuck or at least significantly delayed by trying over and over to follow those docking beams, only to get part-way there and encounter the impassible object. On the other hand, if the robot ignores signals from the home base until it knows that it is in a room where the home base has been previously, successfully acquired, then the robot knows that it can follow the docking beams to a successful docking event.

As claimed, Roomba can clean for about two hours on a single charge. Roomba returns and connects to the charger by itself when the battery power is low. It accomplishes this using the infrared receiver on its front bumper. When the battery power gets low, the vacuum starts looking for the infrared signal emitted by the charger. Once it finds the charger, the Roomba follows the signal and docks itself to the charger.

1.2. AI Real-time Path Planning Algorithm

This paper is mostly concerned with the robot's navigation mechanism. Path planning is a major topic in the field of mobile robot navigation. Autonomous mobile robots are used in various applications such as in automatic freeway driving, cleaning of hallways, exploration of dangerous regions, etc. These applications demand robust and adaptable methods for path planning.

Path planning can be divided into two categories, one is global path planning, where there exists a priori knowledge of the complete working area; and the other is local path planning, where the working area is uncertain. Global path planning includes configuration space method, potential field method and generalized Voronoi diagram. The planning is done off-line, and the robot has complete knowledge of its working area and its path when it starts. Local path planning methods use ultrasonic sensors, laser range finders, and on-board vision system to "perceive" the environment; planning is done on-line [3].

Roomba conducts the local path planning algorithms in most cases as it explores unexplored areas. As we will discuss below, the robot computes a set of heuristic values for the possible steps and choose the best way to go. The calculations are somewhat subjective. The robot would work its way through and figure out the map step after step, and this procedure is complete when the whole area is cleaned. After the local map is built, the robot can conduct the global path planning algorithm to travel on the area that has been explored.

1.3. Finite state machine

This paper will provide a model using the concept of a finite state machine and provide solutions for each state. Some of the solutions are built-in features of the iRobot, and some are new features. But once we can present the problem as a state machine, we can analyze each state separately, which simplifies the analysis as a whole. The states are independent from each other as we will discuss below.

A finite state machine (FSM), or finite state automaton, or simply a state machine, is a model of behavior composed of a finite number of states, transitions between those states, and actions. A finite state machine is an abstract model of a machine with a primitive internal memory [2].

A current state is determined by past states of the system. As such, it can be said to record information about the past, i.e. it reflects the input changes from the system start to the present moment. A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment. There are several action types:

Entry action	performed when entering the state.
Exit action	performed when exiting the state.
Input action	performed depending on present state and input conditions.
Transition action	performed when performing a certain transition.

2. Preliminaries

We assume the robot for this research has the advanced navigation features as the company claims. We also simply the model by considering only parameters that are related to the self-navigation system of the robot.

2.1 Assumptions of Built-in Features

2.1.1 Lighthouse Sensor System

We assume the robot has a fully developed sensor and lighthouse system as the company claims for its higher-end robots. We also assume the user has enough lighthouses according to the dwelling unit floor plan. Normally, we need the same number of the lighthouses as the room numbers, but whenever there is a mostly closed corner shape, we assume the user has placed a lighthouse for the robot to enter that corner. In other worlds, we assume the robot has full access of the house. Our new feature would not introduce in any new mechanism to get around things and reach the spots that were formerly unreachable.

2.1.2. Self-Navigation Algorithm

We assume the robot has fully-developed self-navigation algorithms that work with its sensor and lighthouse system. As we discussed in the introduction, the Roomba is a well-developed AI product, and it is already able to conduct the cleaning task without missing any significant parts of the area. We assume the robot has its own algorithm to calculate the next steps it should take at any given time. For example, the robot would not become confused if there is more than one possible path in front of it. We assume the robot is capable of calculating a set of different heuristic values for a given condition, and that it can decide on the one with the greatest heuristic value during its real-time, local path exploring.

2.2. Approximations and Simplifications

In this paper, we attempt to model a feature that allows the Roomba to resume a task after intermissions of recharging. Generally speaking, we consider the features that are associated with the self-navigation. In other words, we do not consider some of the other problem that might interest the Roomba investigators, such as how to enhance the vacuuming power, how to switch mode by detecting the surface material, and so on.

In this paper, we will present a shortest path searching algorithm on a continuous map, which is a possible solution for one of the states in our state machine. We simplify the floor plan as a grid map. This is a command simplification for this kind of map exploring problem. By doing this, we do not consider too much into the details of the floor plan. In other word, once we divide the room into grids, and a particular grid can be either occupied or not. There are only two states for the grids. If the dividing does not provide a reasonable map, we will have to cut it into more girds. We either sacrifice a little bit of the algorithm performance, or sacrifice a little bit of precision.

3. Problem formulation

3.1 Key Concepts

Before we dive into the problem formulation, we need to clarify two essential concepts for this model. These two concepts are also the logic and reasoning that make our new feature possible to implement. This first one is how does the robot conduct the battery-reserving-mode cleaning. The second one is how to achieve traveling from A to B with the minimal battery consumption.

3.1.1. Cleaning in a Batter-Reserving Mode

3.1.1.1. Battery-Reserving Mode Cleaning.

As the battery goes low, the robot has to decide when to turn around and go back to the docking station. Since the traveling itself costs battery life, the robot does not want to wait long and take the risk of dying on the way home. Taking a long backtrack home is not a good idea either because the traveling is done on areas already cleaned. In some sense, the battery life used for traveling from place to place is a waste.

We consider the lighthouse works as physical separators. After passing each lighthouse, the robot calculates the time consumption to travel home from that lighthouse. For each lighthouse it is a new start, a new calculation of the local distance from the current lighthouse. Therefore, when the battery starts to go low, the robot already has data of the time needed to travel home from the last lighthouse. The decision to make is how far away it should go in the current room in order to reserve enough battery to go home. If that distance covers the whole room, the robot recursively does the same calculation for the next room it enters.

3.1.1.2. Simple Example

As we discussed in the preliminary section, the robot has its method to calculate the next most

attractive step. It calculates a series of heuristic values for the next steps and takes the best one. We observe this somewhat follows the keep left/right algorithm. So let us just draw the tracks as shown below. The path that the robot takes does not necessarily need to be the same as the figure but we are sure that it tends to go towards the unexplored areas if possible.

In this mode, the robot still processes the regular heuristic value calculation. The difference is it refuses to take the steps that lead it be too far away from the last lighthouse it passed. Within the robot database, the robot knows how long it will take to go home from the last lighthouse. Then, it can calculate how far it can travel within the current area. When the battery life passes a certain level, the robot turns on the last lighthouse to define a region and sets itself to the battery-reserving mode.

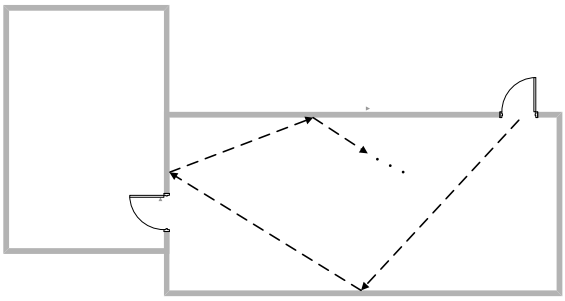

Figure 4. The Original Tour

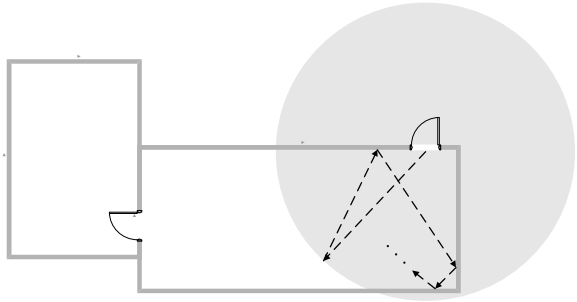

Figure 5. The Tour in Battery-Reserving Mode

3.1.2. Shortest Paths on Explored Maps

As we will discuss later, there are several cases in which the robot needs to travel from one place to another on an explored area of the map, and the purpose of these movements is simply moving to the other part of the map. We can define "explored area" as "already covered area," "already cleaned area," and so on.

This scenario happens in difference cases: when the robot needs the resume its cleaning after charging, when the robot needs to go home after realizing the battery is low, and when the robot finishes one room and needs to go back to the last lighthouse it passed. As we can see, in all of these scenarios the robot travels on a known portion of the map.

Since the area is already cleaned, we do not want to spend too much time in traveling in this region because it would be a waste of battery life. We need to find the shortest path as the solution of this problem. A* algorithms serve this purpose.

3.1.2.1. A* Algorithms

In computer science, A* is a best-first, graph search algorithm that finds the least-cost path from a given initial node to one goal node. This is a variant of Dijkstra's algorithm, which is more suitable for a continuous grid-based map [4].

$g(x)$: the actual shortest distance traveled from initial node to current node

$h(x)$: the estimated (or "heuristic") distance from current node to goal

$f(x)$: the sum of $g(x)$ and $h(x)$

3.1.2.2. Simple example of A* Algorithms

Let's put this implementation into a simplified example. We have the robot sitting at position A, and the destination is position B. The destination could be the docking station, or where the robot stopped its work for the last trip, and it is time to go back and resume cleaning. We have two obstacles in between the robot and its desired location. If A and B are located in two different rooms, we can think of the obstacles as the walls. If A and B are located in the same room, we can think about the obstacles as the furniture.

The grids on this map are already drawn as the robot visited this area the last cleaning session. As we discussed in the preliminary, we consider the robot has a short term memory of the room layout. This means the robot remembers the visited area within a day or two. It allows the robot to resume the cleaning work after intervals of charging. But after a couple of days, the robot will always regenerate this map as it goes through it and cleans the area.

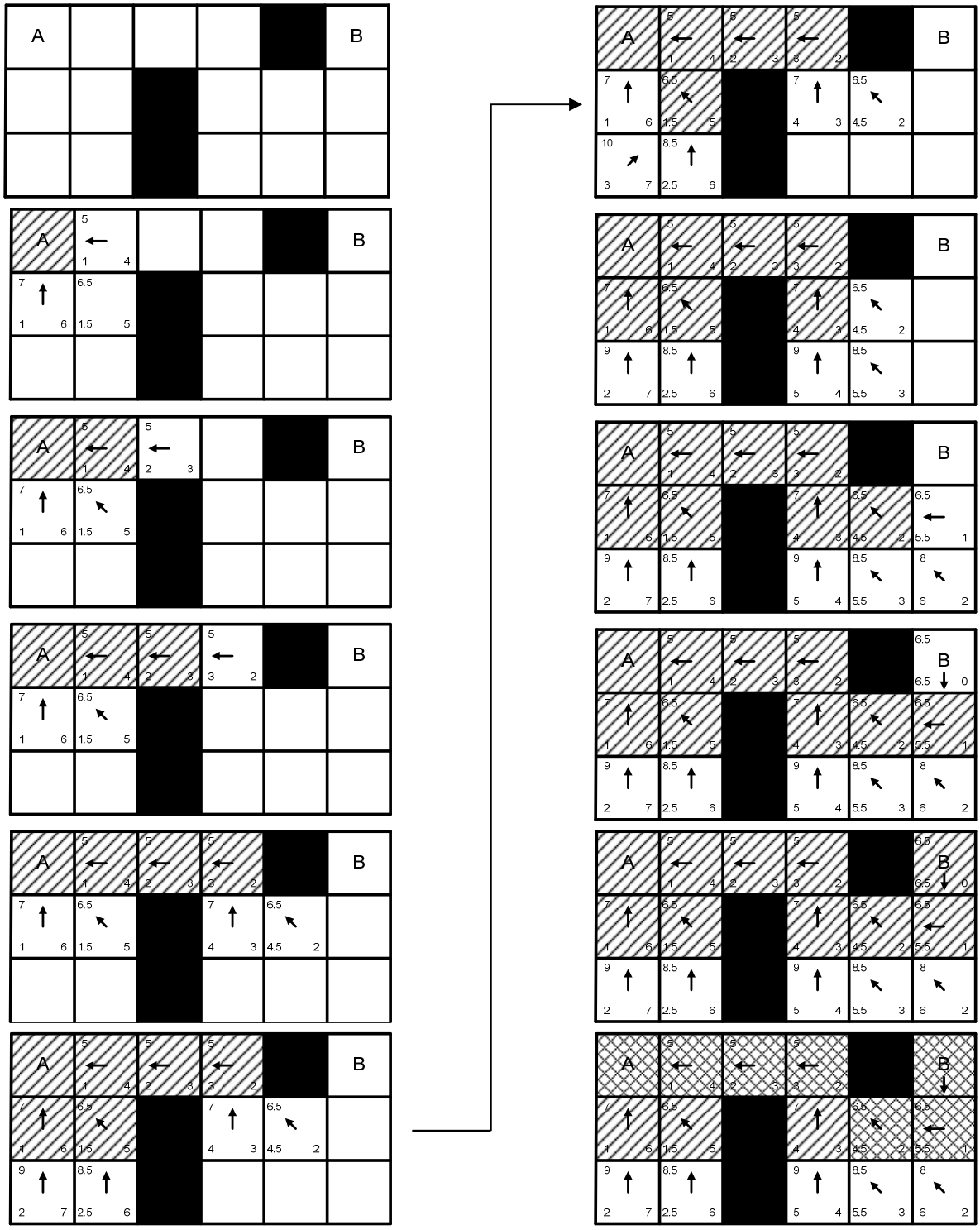

Figure 6. Execution of A* Algorithm

In Figure 6, the number on the top-left is the current $f(x)$ for this grid, the number on the bottom-left is the local $g(x)$ for this grid, and the number on the bottom-right is the estimated distance from this node to the destination $h(x)$. For this implementation, we used Manhattan's method, which finds the shortest path on the grids without considering the obstacles and without cutting the blocks diagonally. This is just one way to estimate, there are other ways to make the guess and adjust the actual distances as the algorithm executes.

The figures also used shading to represent the closed set, which includes the nodes that are already explored. The open set is shown as nodes without shading. They contain function values and parent pointers. The nodes in the open set are waiting to be examined. They are stored in a priority queue based on their objective function value, $f(x)$. They are polled out one after another and placed into the closed set. At the same time, we add their directly connected neighbors that are not in the open set into the open set, and update their $g(x)$ and $f(x)$ values. The figures use crosshatch to represent the final path. Once we reach the destination, we simply trace back the parent pointer from the destination node.

3.2. Problem Configuration and Setup

Hereon we formulate our problem as a state machine that consists of five states. The entry action, exit action, transition action, and state performances are described in detail for each of the states below.

3.2.1. State 1: Charging

Entry action: There are two ways to start this state. The robot can be manually placed on the charger. The robot can execute its self-charging functionality when it comes back to the home base.

1. State starts manually
2. State starts as self-charging

Exit action: There are three possible states after exiting this state. The robot may start from beginning a new round of house cleaning. The robot may resume its cleaning work which has not been finished at the last mission. The robot may remain on the charger after terminating the charging process. The third case would happen if the robot had finished its cleaning work during the last mission while it has not received any new job request. It is also possible that it is not time for another round of cleaning yet, if the timer system is used for automatic scheduling.

1. Enter state 2 → cleans in the regular mode
2. Enter state 4 → travels from A to B
3. Enter state 6 → terminates state machine

Input action: The task for this state is clear—charging. There are two operations involved in order to properly perform this task. If the battery is not fully charged, start charging. If the battery is fully charged, the charging process automatically shuts down.

Generally speaking, there is only one condition that determines the performance of this state: whether or not the robot is fully charged.

1. Condition check: battery is low
   → Performance 1: start charging
2. Condition check: battery is full
   → Performance 2: terminate charging

Transition action: There are two entry actions and three exit actions. There are five transition actions associated with these performances. For entry action 1, it occurs with human interference and it is a possible way to start this state. For entry action 2, it occurs when the robot comes back to its home base and starts its self-charging. For exit action 1, it occurs when the robot starts a new round of house cleaning. It starts from the first room where the docking station is. For exit action 2, it occurs when the robot needs to go back to where it was the last time cleaning, and starts from that point on to clean the rest of the house. For exit action 3, it occurs when the house is completely cleaned after the last trip. The state machine is done execution for this round.

1. Entry action 1 → human interference
2. Entry action 2 → robot comes back
3. Exit action 1 → robot starts new task
4. Exit action 2 → robot resumes last task
5. Exit action 3 → robot stands by

3.2.2. State 2: Cleaning in Regular Mode

Entry action: There are three ways to start this state. The robot starts a new round of cleaning work from the first room after charging. The robot resumes its cleaning work from the last trip after charging. The robot enters a new room and continues its cleaning work.

1. States starts after charging
2. State starts after finding the right position
3. State starts when entering a new room

Exit action: There are three possible states after exiting this state. The robot may be finishing up with one room and sends out a request for the next closest lighthouse signal, therefore entering state 5. This may be the last room to clean. The robot may get ready to head back to its home base—the charging station, therefore entering state 4. During the cleaning within this room, the robot may realize that its battery is running low. It may set itself to a battery-reserving-cleaning mode, which is state 3.

1. Enter state 5 → requests for lighthouse signal
2. Enter state 4 → travels from A to B
3. Enter state 3 → cleans in the battery-reserving mode

Input action: The task for this state is clear—cleaning in a room. There are several operations involved in this state in order to properly perform the task.

Perform the regular cleaning. According to our preliminaries and the iRobot built-in feature, the robot has its corresponding algorithm designed to perform this task. It should always take the next most reasonable step to explore this unexplored area with a certain number of floor coverage.

Determine if it is the finishing point of the room. According to our preliminaries and the iRobot has its hardware and software working together to detect the stopping point of the certain room. The sensor system keeps telling the robot its current position in the room. The software constantly computes the overall coverage of this area and tells the robot if it is time to stop.

Detect the battery storage in real-time. This is a routine operation for a variety of electronics. We assume the procedure of measuring the current battery storage takes almost no time, and the robot would constantly repeat this operation to keep alert if the battery is going low.

Determine if this is the end of this round of house cleaning. According to our preliminaries and the iRobot built-in feature, it determines the coverage of rooms in the house by keep tracking how many lighthouses it has passed. If there is no more lighthouse to pass and the current room in finished, the robot knows that it hit the end point of this round of cleaning. Therefore, it turns around and is ready to go home.

Therefore, generally speaking, there are three conditions that the robot is constantly checking: what is the next most attractive step to take for the cleaning/exploring purpose; is it the finishing point of this room; what is the current battery condition. There is one condition that the robot needs to check at the end of the state: is this the last room to clean.

1. Constantly
   → Performance 1: computing the next most attractive step
2. Constantly
   → Performance 2: computing the current coverage status
3. Constantly
   → Performance 3: detecting the current battery condition
4. Condition check: this room is finished
   → Performance 4: determine if the whole house is finished

Transition action: There are three entry actions and three exit actions. There are six transition actions associated with these performances. For entry action 1, it occurs after the charging terminates. For entry action 2, it occurs when the robot enters a new room that is unexplored. For entry action 3, it occurs when the robot decides this is where it left in the last trip and the cleaning work is not finished yet. For exit action 1, it occurs when the robot finishes one room and ready to explore more unexplored area and continue this round of cleaning work. It, therefore, sends out the request for entering another room. For exit action 2, it occurs when the robot finishes one room and detected that this is the last room to clean. For exit

action 3, it occurs when the robot decide the battery is getting low and ready to reset itself into a battery-conserving-cleaning mode.

1. Entry action 1 → robot charging terminates
2. Entry action 2 → robots enters a new room
3. Entry action 3 → robot finds where to resume cleaning
4. Exit action 1 → robot finishes the room
5. Exit action 2 → robot finishes task
6. Exit action 3 → robot detects low battery

3.2.3. State 3: Cleaning in Battery-Reserving Mode

Entry action: There is only one way to start this state. That is from state 2. The robot detects the battery storage constantly during the regular cleaning. When it detects that the battery is getting low, it is possible to perform a little bit more cleaning work but cannot afford going any further from its docking station.

State starts after cleaning with regular mode

Exit action: There are two possible states after exiting this state. Robot needs to go back to the home base—the charging station, therefore entering state 4. Although there is only one following state in this case, there are two different cases when the robot leave this state and enter state 4. We will discuss them in the transition action section. The robot may finish the assigned area before the battery becomes critically low. Then, it may also request for the next lighthouse signal to lead it into a new area.

1. Enter state 4 → travels from A to B
2. Enter state 5 → requests the lighthouse signal

Input action: The task for this state is to clean the room in a battery-reserving-cleaning mode. There are several operations involved in this state in order to properly perform the task.

Perform the battery-reserving-mode cleaning. According to our preliminaries and the iRobot built-in feature, the robot has its corresponding algorithm designed to calculate the next most attractive step. It should always provide a series of heuristic values for the following possible steps. In this mode, the robot refuses to take the steps that lead it farther than a certain distance away from the last lighthouse.

Determine if it is the finishing point of the room. The robot is still doing this calculation in this mode of cleaning. According to our preliminaries and the iRobot has its hardware and software working together to detect the stopping point of the certain room. The sensor system keeps telling the robot its current position in the room. The software constantly computes the overall coverage of this area and tells the robot if it is time to stop.

Detect the battery storage in real-time. As part of the routine, the battery life detection is still executing in this mode. The difference here is battery condition may switch from low to critically low during the state.

Determine if this is the end of this round of house cleaning. If the robot hit the finish point of current room while it has not turned switch to state 4, the robot still does this calculation in this mode of cleaning. According to our preliminaries and the iRobot built-in feature, it determines the coverage of rooms in the house by keep tracking how many lighthouses it has passed. If there is no more lighthouse to pass and the current room in finished, the robot knows that it hit the end point of this round of cleaning. It, therefore, turns around and ready to go home.

Therefore generally speaking, there are three conditions that the robot is constantly checking: what is the next most attractive step to take for the cleaning/exploring purpose; is it the finishing point of this room; what is the current battery condition. There is one condition that the robot needs to check at the end of the state: is this the last room to clean.

1. Constantly
   → Performance 1: computing the next most attractive step within a certain distance
2. Constantly
   → Performance 2: computing the current coverage status
3. Constantly
   → Performance 3: detecting the current battery condition

4. Condition check: this room is finished
   → Performance 4: determine if the whole house is finished

Transition action: There are one entry actions and two exit actions. But there are four transition actions associated with these performances. For entry action 1, it occurs when the robot realizes that the battery is low and it is time to switch to the battery-cleaning mode. For exit action 1, it occurs in two scenarios, when the robot finishes the rest of the house within this state and it turns back goes home; when the robot detects that the battery is into a critically low level and it is time go home with no delay. For entry action 2, it occurs when the robot finishes this room within this mode and request for the next lighthouse signal.

1. Entry action 1 → robot detects low battery
2. Exit action 1 → robot finishes the task
3. Exit action 1 → robot detects the battery is critically low
4. Exit action 2 → robot finishes the room

3.2.4. State 4: Traveling from A to B

Entry action: There are four ways to start this state. The robot needs to resume its cleaning work from the last trip after charging. The robot receives the lighthouse signal and ready to leave the current room and enter a new room. The robot finishes a round of house cleaning work and ready to go back to the charging station. In this last case, it may be during the regular cleaning or within the battery-reserving-mode.

1. States starts after charging
2. State starts after receiving the signal
3. State starts after finishing the task within the regular mode
4. State starts after finishing the task within the battery-reserving mode

Exit action: There are three possible states after exiting this state. The robot may start cleaning, dock, or start charging. The first case contains the following states. It may resume cleaning with full battery power in the regular mode, state 2. It may enter a new room and resume the cleaning in the regular mode, state 2. It may enter a new room and resume cleaning in its battery-reserving mode state 3. The second case contains one following state, state 1—charging state.

1. Enter state 2 → cleaning in the regular mode
2. Enter state 3 → cleans in the battery-reserving mode
3. Enter state 1 → charging

Input action: The task for this state is traveling with the shortest path. We discussed the algorithm for this path planning in the previous section. There are two operations involved in order to properly perfume this task. The task is straightforward—traveling. The first operation is calculating the shortest path from A to B using A* algorithm. The second operation is requesting lighthouse navigation constantly.

Perform the calculation and determine the shortest path from A to B. As we discussed this can be done using A* algorithm, which compute within a grid-based continuous map.

Request the lighthouse navigation. The robot keeps a list of the lighthouses it passed and request the navigation from them one after the other. The lighthouse set the next destination as the robot travels.

Therefore generally speaking, there are three conditions that the robot is constantly checking: where it is on the map and how far it is from the active lighthouse. There is one calculation that the robot needs to compute: based on the stored map, what is the shortest path to the active lighthouse.

1. Constantly
   → Performance 1: where it is on the map
2. Constantly
   → Performance 2: how far it is from the active lighthouse
3. Condition check: reaches the lighthouse
   → Performance 3: request for the next lighthouse.
4. Condition check: receives the next signal
   → Performance 4: calculate the shortest path from current position to that lighthouse.

Transition action: There are four entry actions and three exit actions. There are seven transition actions associated with these performances. For entry action 1, it occurs after the charging terminates. For entry action 2, it occurs when the robot finishes the current room and ready to enter a new room that is unexplored. For entry action 3 and 4, it occurs when the robot finishes this round of house cleaning, and ready to go back to the charging station and charge itself. For exit action 1 and 2, it occurs when the robot reaches a certain place where it needs to resume the cleaning work. For exit action 3, it occurs when the robot goes back to the charging station and ready to perform self-charging.

1. Entry action 1
   → robot charging terminates
2. Entry action 2
   → robots finishes the current room
3. Entry action 3, 4
   → robot finished the whole house
4. Exit action 1, 2
   → robot finds the place to resume cleaning work
5. Exit action 3
   → robot reaches the charging station

3.2.5. State 5: Requesting the Lighthouse Signal

Entry action: There are only two ways to start this state. That is from state 2 and state 3. The robot detects that it reaches the finish point of a current room. It sends out request to go to the next room if there is any. It may happen during the regular cleaning or during the battery-reserving mode.

1. States starts after finishing cleaning a room with sufficient battery
2. States starts after finishing cleaning a room within the battery-reserving mode

Exit action: There is only one possible state after exiting this state. Robot needs to leave the current room, which is finished. To perform this task, the robot enters state 4. It takes the shortest path from its current position to the active lighthouse, which will lead it to the next unexplored area.

Enter state 4, travels from A to B

Input action: The task for this state is simple—sending out request to the next lighthouse. As the users' observance, the robot freezes itself for a second when it reaches the finishing point of a certain area and ready to clean a new room. The task of this state is performed during that time interval. The robot communicates with the lighthouse. After it receives the navigation signal, the task of this state is finished. The robot enters state 4, when it takes the shortest path to move to the active lighthouse.

Perform the communication with the lighthouse. According to our preliminaries and the iRobot built-in feature, the robot has its corresponding algorithm to decide what the next lighthouse is to turn on. It communicates with only that one lighthouse.

1. Constantly
   → Performance 1: is this the finishing point of the current room
2. Condition check: this room is finished
   → Performance 2: request to turn on the next lighthouse

Transition action: There are two entry actions and one exit actions. But there are only two transition actions associated with these performances. For entry action 1 and 2, it occurs when the robot realizes that this is the end of the current room. For exit action 1, it occurs when the robot receives the navigation signal as the response of its request.

1. Entry action 1, 2
   → Robot finishes cleaning the current room
2. Exit action 1
   → Robot receives the lighthouse signal

3.3. State machine

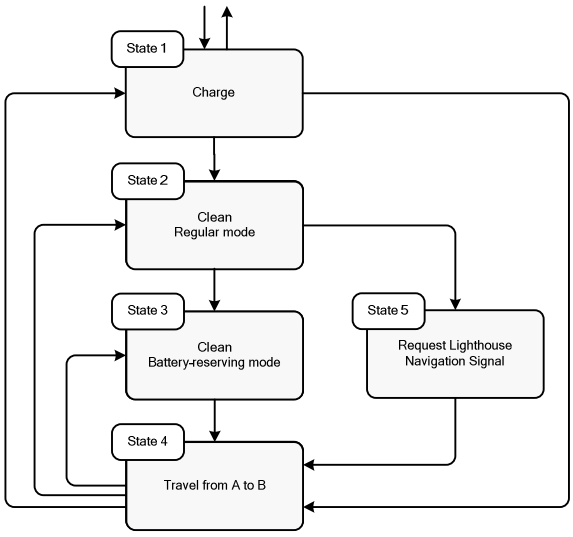

Figure 7. Model—The State Machine

So far, we categorized all the performances into different states. Each state has its entry state/states and exit state/states. We also established the state performances with their exiting conditions, which tell us when to leave this current state and which next state to enter.

Based on the definition of the state machine, we know we can conquer the problem as a whole by seeking solutions for each of the state. The state performances should be independent from other states. Let's look into the solutions for our five states.

3.3.1. Solution for State 1—Charging

Apparently, it is safe to assume the robot has a solution for this state because it is part of the nature of all most all electronics.

3.3.2. Solution for State 2—Cleaning in the Regular Mode

As we discussed in the introduction section and the preliminary section, we assume the robot has fully developed self navigation algorithms that work with its sensor and lighthouse system. The robot is able to conduct the cleaning task without significantly missing any parts of the area. The robot calculates and constantly provides a set of heuristic values for the next possible steps. It compares and takes the most attractive step as it explores the area.

The robot calculates:

$$h(step_1), h(step_2), h(step_3), \dots, h(step_n)$$

Heuristic value set:

$$v_1, v_2, v_3, \dots, v_n \text{ where } h(step_n) = v_i$$

Suppose we have:

$$v_1 < v_2 < v_3 < \cdots < v_n$$

We learn that $h(step_n)$ provides the largest heuristic value, and $step_n$ is, therefore, the best guess according to the robot.

3.3.3. Solution for State 3—Cleaning in the Battery-Reserving Mode

As we discussed in the key concepts section of the problem formulation, this is solved by refusing the steps that lead the robot farther than a certain distance from the last lighthouse. In this mode, the robot still tries to explore the scheduled area according its navigation algorithms. However, once the robot realizes that the battery is low to a certain level, it requests the last lighthouse to set a distance limitation and refuse to take the steps take the robot outside of this area.

The robot calculates:

$$h(step_1), h(step_2), h(step_3), \dots, h(step_n)$$

Heuristic value set:

$$v_1, v_2, v_3, \dots, v_n \text{ where } h(step_n) = v_i$$

Suppose we have:

$$v_1 < v_2 < v_3 < \cdots < v_n$$

Suppose we also have $v_2, v_4,$ and v_n v_2, v_4 that takes the robot out of the restricted range. In this mode, the robot would therefore pick $step_{n-1}$ as the next most attractive step. Although based on its original navigation algorithms $step_n$ has a even higher heuristic value, the robot refuses to take it once it is in this battery-reserving mode

3.3.4. Solution for State 4—Travel from A to B

As we discussed in the key concepts section of the problem formulation, this is solved by implementing A* algorithm. In this mode, the robot is either going home from the end point of its current work, or resuming the last round of cleaning. In either case, the robot needs to travel from A to B directly without any delay. Also, since the area that the robot travels on is already explored. The robot has built a temporary map for it. We can apply A* algorithm to compute the shortest path to get from A to B.

3.3.5. Solution for State 5—Request for Lighthouse Navigation

Apparently, it is safe to assume the robot has a solution for this state because it is one of the most important features that Roomba has. This is also the fundamental feature that the robot conducts the self navigation to cover the whole house without human interference.

4. Conclusion and Remarks

In this paper, we modeled a new feature for Roomba—the iRobot. It provides a possibility for the robot to resume its work after intervals of charging. This paper also modeled the robot's working behaviors under different battery storage conditions.

The model was constructed under a series of assumptions and simplifications. We assumed the robot in this research has a full self-navigation and sensor system as the company claimed. We considered most of the possible factors associated with the navigation mechanism while we did not consider other factors that might also interest customers and investigators.

The problem was formulated as a finite state machine. We presented the entry actions, exit actions, performances, and transition actions for each state. We further analyzed the solutions for each state's performances. The operations for each state could be achieved either by the robot's built-in functionality or the novel features that we presented.

The definition of the state machine was provided, but the model was not meant to be used directly. Instead it is meant to serve as a modeling reference for future generations of Roomba development.

5. References

[1] http://home.howstuffworks.com

[2] http://forums.irobot.com/irobothome/board

[3] I.J. Nagrath, Laxmidhar Behera, K. Madhava Krishna, and K. Deepak Rajasekar, *Real-time navigation of a mobile robot using kohonen's topology conserving neural network*

[4] http://en.wikipedia.org/wiki/A*_search_algorithm