Minimalist Robot Navigation and Coverage using a Dynamical System Approach

Tauhidul Alam[†], Leonardo Bobadilla[†], Dylan A. Shell[‡]
[†]School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
Email: {talam005, bobadilla}@cs.fiu.edu
[‡]Department of Computer Science and Engineering
Texas A&M University
College Station, TX 77843, USA
Email: dshell@cse.tamu.edu

***Abstract*— Equipped only with a clock and a contact sensor, a mobile robot can be programmed to bounce off walls in a predictable way: the robot drives forward until meeting an obstacle, then rotates in place and proceeds forward again. Though this behavior is easily modeled and trivially implemented, is it useful? We present an approach for solving both navigation and coverage problems using such a bouncing robot. The former entails finding a path from one pose to another, while the latter combines different paths over desired locations. Our approach has the following steps: 1) A directed graph is constructed from the environment geometry using the simple bouncing policies; 2) The shortest path on the graph, for navigation, is generated between either one given pair of initial and goal poses or all possible pairs of initial and goal poses; 3) The optimal distribution of bouncing policies is computed so that the actual coverage distribution is as close as possible to the target coverage distribution. Finally, we present experimental results from multiple simulations and hardware experiments to demonstrate the practical utility of our approach.**

Keywords: Navigation; Coverage; Minimalist robot; Cell-to-Cell mapping; Dynamical system.

I. Introduction

The problem of navigation is finding a path between an initial pose and a goal pose. The coverage problem of the environment is passing over all locations of interest using a robot. These problems are important for many applications, such as surveillance, vacuum cleaning, environmental decontamination, demining, and searching and rescue. The motivation of our work is to find solutions to common robotic problems using a minimalist robot.

Minimal sensing robots have been used to solve different tasks such as localization [1], navigation [2] [3] [4], and mapping [5]. In sensor-denied environments, some extrinsic sensors such as GPS will not work properly, and compass readings can be disturbed by electromagnetic fields. In some cluttered environments, vision-based perception can also be ineffective and expensive. In these cases, we rely on inherent sensing robot as it avoids using error-prone sensors and actuators. So, for solving navigation tasks in an environment, our robot uses only limited linear and angular sensing. In the case of coverage, probabilistic coverage strategies can be useful in adversarial environments [6]. So, we consider the nondeterministic angular rotation of the robot for solving the coverage task.

Although sensors are cheap now but the overuse of sensors requires powerful computation system and abundant memory inside the robot. Instead, we use a little-brained robot that can feed minimal sensor outputs directly to motors. So, in this work, we investigate two robotics tasks by proposing *navigation* and *coverage* for a robot that has only a contact sensor and a clock, after its localization. Here we apply a dynamical system method called *generalized cell-to-cell mapping* (GCM).

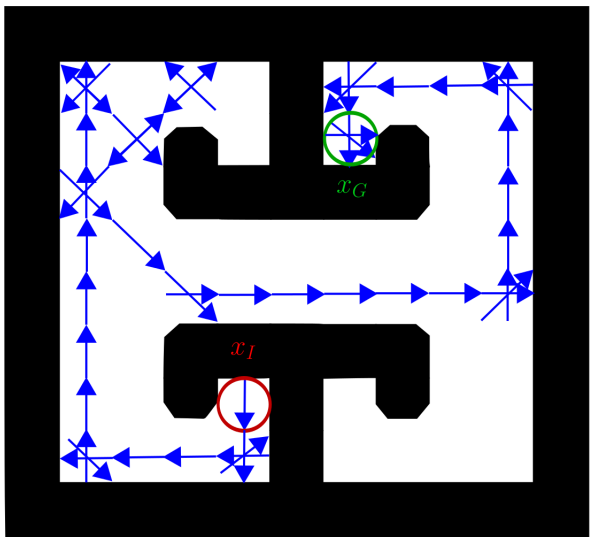


Fig. 1. A navigation plan generated by our algorithm between the initial configuration (red circled location facing South) and the goal configuration (green circle location facing East) using a bouncing robot that bounces with angles 45°, 90°, and 135°.

The robot moves in a known polygonal environment and has a simple behavior which is characterized by a set of angles with an error range. We term this set of angles as the set of *bouncing angles*. The robot moves straight in the environment and once it discovers walls, by driving into them, it rotates counterclockwise by a bouncing angle with respect to its current moving direction, including the error in rotation.

Our work makes the following contributions:

- We propose an algorithm based on the generalized cell-to-cell mapping method to find the minimum navigation plan for a minimalist robot between an initial and a goal configuration in the environment.
- We generate all minimum navigation plans between all possible initial and goal configuration pairs in the environment.
- We develop a method for finding a probability distribution of bouncing policies for the best possible coverage of the environment with respect to a target coverage distribution.

The remainder of our work is organized as follows. Section II reviews the related literature of minimalist robot navigation and coverage. Section III defines the robot model, explains cell-to-cell mapping, and gives formal problem statements. In Section IV, we describe the method of our work in detail. Then Section V illustrates our simulation results and physical implementations of our work. Finally, we conclude in Section VI with discussion and a preview of our future work.

II. Related Work

This section discusses related literature first on navigation of minimalist robots and secondly on the simple robot coverage.

A. Robot Navigation

Early works on landmark-based robot navigation include [7] [8]. In these works, authors consider that the robot goes from one landmark to another with the explicit sensing of landmarks. However, in our work, we use the geometric description of the environment for navigation instead of the explicit sensing of landmarks.

More recent works on the robot navigation are belief roadmaps [9], randomized belief space trees [10], and feedback-based information roadmaps [11] which consider the robot's uncertainty in its navigation. In our work, we have used a simpler robot model compared to those used in these other works. The works [2] [3] [4] use a robot model which is much closer to ours. In these works, authors use a robot equipped with a compass and a contact sensor whereas we use a robot equipped with a clock and a contact sensor. In their work, the robot can orient itself using the compass in the desired direction relative to a global reference frame which makes their robot stronger than our robot. Our robot instead follows a simple bouncing behavior to get to the goal configuration from its initial configuration. While they do not consider the weight of the navigation path, we minimize it.

B. Robot Coverage

The problem of coverage by a mobile robot has been investigated in different studies. In one survey on coverage path planning [12] studied where the approaches are evaluated based on whether they can be used online or offline and on the type of environments they can handle. In [13], an online topological coverage algorithm for mobile robots is presented that uses the detection of landmarks. Again, explicit sensing of landmarks is required so that the area of the environment will remain uncovered where no landmarks are available. Also their method cannot find the critical points of concave landmarks as obstacles. In [14], a coverage solution for mobile robots is presented which finds critical points of obstacles in unstructured environments and gives the entry and exit critical points for each obstacle.

In [15], the authors propose fast coverage of the environment based on the unpredictable trajectory of a mobile robot with the use of a Logistic map and provide a chaotic random bit generator for a time-ordered succession of future robot locations. However, in their work, some parts of the environment stay uncovered. In an adversarial setting, the probabilistic method can optimally cover the environment and maximize the chances of detecting adversaries [16]. Hence, in this work, we are interested in finding the optimal distribution of the bouncing policies used by our robot to get our intended coverage of the environment.

III. Preliminaries

In this section, we introduce our robot model, describe cell-to-cell mapping with required notation and definitions, and formulate the navigation and coverage problems of interest.

A. Robot Model

We consider a polygonal *workspace* $\mathcal{W}$ which contains an inaccessible region (*obstacle region*), denoted $\mathcal{O} \subset \mathcal{W}$. The robot can move in a subset of workspace that excludes the obstacle region which is called the *free space*, $E = \mathcal{W} \setminus \mathcal{O}$. Let $\partial E \subset E$ be the boundary of E.

In the environment E, we have a differential drive robot modeled as a point robot. We assume that the robot has a map of E and a finite set of angles Φ by which it can rotate reliably. This robot is equipped only with a clock and a contact sensor.

The robot travels forward in the environment and records the number of *steps* using the clock as a *limited linear odometer* which we call a *pedometer*. It continues the forward motion until its bump sensors bump into the boundary of the environment ∂E. After bumping at ∂E, the robot uses the clock again as a limited functional *angular odometer* to rotate counterclockwise, by one bouncing angle $\phi \in \Phi$ including the error range $\pm\epsilon$, from the current direction of the robot. Then, it travels forward until it bumps at ∂E again and repeatedly follows this simple behavior.

B. Cell-to-Cell Mapping

The configuration space of the robot is $X = E \times S^1$, where S^1 is the set of directions in the unit circle that represents the robot's orientations. Let $x \in X$ denote the configuration of the robot, in which $x = (x_t, y_t, \theta)$ where (x_t, y_t) represents its position and θ provides its orientation. Let $\mathcal{A}(x) \subset \mathcal{W}$ be a closed set that defines the robot. The obstacle region $X_{\text{obs}} \subset X$ in the configuration space [17] is defined as

$$X_{\text{obs}} = \{x \in X | \mathcal{A}(x) \cap \mathcal{O} \neq \emptyset\} \tag{1}$$

and the leftover configurations constitute the free space which is denoted $X_{\text{free}} = X \setminus X_{\text{obs}}$.

The configurations of the robot are limited to a subset of the whole configuration space. This subset of the configuration space is denoted by X_{free}. We discretize X_{free} by dividing it into equal sized 3-dimensional box cells. Let N be the total number of cells. This discretized configuration space is also called a *cell configuration space*. Each cell represents an indivisible unit in the cell-to-cell mapping system entity. The configuration of the system is indexed by a cell index $z \in \{1, \ldots, N\}$. Let $Z = \{1, \ldots, N\}$ denote the set of all cells.

In the cell-to-cell mapping [18], the system dynamics are described by

$$p(n+1) = \mathcal{P}p(n) \text{ or } p(n) = \mathcal{P}^n p(0) \tag{2}$$

the above dynamical system evaluation is called the generalized cell-to-cell mapping (GCM) that creates finite Markov chains where $\mathcal{P}$ is the one-step transition probability matrix and $\mathcal{P}^n$ is the n-step transition probability matrix, $p(0)$ is the initial probability distribution vector over the cell configuration space, and $p(n)$ is the n-step probability distribution vector over the same configuration space. Let $p_{ij}^{(k)}$ denote the k-step transition probability from cell i to cell j and be (i,j)-th element of $\mathcal{P}^{(k)}$. If it is possible through the mapping to go from cell i to cell j, we say that cell i leads to cell j, symbolically $i \Longrightarrow j$. Analytically, cell i leads to cell j if and only if there exists a positive integer k such that $p_{ij}^{(k)} > 0$. The cells i and j

are said to communicate if and only if $i \Longrightarrow j$ and $j \Longrightarrow i$ which is denoted by $i \Longleftrightarrow j$.

To make our work complete, here are some important definitions of the GCM method. More definitions and descriptions can be found in [19] [20] [21].

Definition 2.1: (Persistent Cell) A cell z is called a persistent cell if it has the property that when the system is in z at a certain moment, it will return to z at some time in the future.

Definition 2.2: (Transient Cell) A cell that is not persistent is called a *transient cell*. It leads to a persistent group in some number of steps.

Definition 2.3: (Persistent Group) A set of cells that is closed under the mapping is said to form a persistent group if and only if every cell in that set communicates with every other cell. Each cell belonging to a persistent group is called a persistent cell.

C. Problem Formulation

We define a finite *observation space* Y which is a set of discrete observations. The robot observes $\{0\}$ if it moves forward, and observes the angle of rotation $\phi \in \Phi$ if it rotates, therefore, the observation space is $Y = \{0\} \cup \Phi$.

Based on the resolution and the clock time between two bumps, our robot measures the linear distance traversed by the number of *steps* up to some quantization error. From this measurement, robot receives a string of 0s as a stream of observations. During a bump event, the robot measures the bouncing angle ϕ using the clock time, resets the clock, and receives the value of ϕ as a discrete observation. Therefore, the sequence of observation of our robot, called the *observation string* $\tilde{y}$, is encoded as a string of 0s interspersed by a value of ϕ.

Let $x_I \in X_{\text{free}}$ be the initial configuration of robot $\mathcal{A}$, and $x_G \in X_{\text{free}}$ be its desired goal configuration. We assume that $\mathcal{A}$ knows x_I and x_G for the navigation between x_I and x_G, and rotates reliably. In this context, a single query (x_I, x_G) navigation problem is formulated as follows:

Navigation Problem 1: Finding Minimum Plan

Given an environment E, a set of bouncing angles Φ, an initial configuration x_I, and a goal configuration x_G, find the observation string $\tilde{y}$ as a minimal navigation plan for the robot from the shortest path involving the minimum number of bouncing angle changes along the path, if one or more paths exist.

Each bouncing angle represents a separate bouncing policy for the robot. For answering multiple (x_I, x_G) navigation plan queries for the robot, we can extend the single query navigation plan problem by finding all possible shortest paths between the initial and goal configuration pairs using the given set of bouncing angles. As such, the multiple queries navigation problem is formulated as:

Navigation Problem 2: Generating All Minimum Plans

Given an environment E, a set of bouncing angles Φ, generate all observation strings as minimum navigation plans for the robot from all possible shortest paths for all (x_I, x_G) pairs in the X_{free}, involving the minimum number of bouncing angle changes along these paths.

In the coverage problem scenario, $\mathcal{A}$ does not need to know about x_I and x_G. However, $\mathcal{A}$ has a target coverage distribution over E which is denoted by b and its bouncing angle has a small error range. To combine the bouncing policies for coverage, the best solution is to find the optimal bouncing policy distribution of the robot based on the long-term robot's behavior resulting from the application of these policies. Let α be the bouncing policy distribution of the robot. So, the coverage problem is formulated as:

Coverage Problem: Finding Optimal Bouncing Policy Distribution

Given an environment E, a set of bouncing angles Φ, an error range $\pm\epsilon$, and a target coverage distribution b, find the optimal bouncing policy distribution α to get as close coverage as possible to b.

IV. Methods

This section describes the methods that solve the navigation and coverage problem introduced in Section III.

A. Finding Roadmap and Minimum Navigation Plan

Let a topological graph $\mathcal{G} = (V, E)$ be a weighted, directed graph and the weight function $w : E \to \mathbb{N}^+$ assign the nonnegative edge weight. Each vertex $v \in V$ represents a configuration (cell) $x \in X_{\text{free}}$ and each edge $(u, v) \in E$ where $u, v \in V$, represents a configuration transition from $x \in X_{\text{free}}$ to $x' \in X_{\text{free}}$. This topological graph is also called a *roadmap*. If any path exists between the initial configuration x_I and the goal configuration x_G on $\mathcal{G}$, then there will be one or more paths among (x_I, x_G) ordered pairs for the set of bouncing angles Φ. Let a shortest path of the robot $\mathcal{A}$ be $\tau : [0, 1] \to X_{\text{free}}$ such that $\tau(0) = x_I$, $\tau(1) = x_G$. After the robot's bump event, if the bouncing angle is preserved, the weight of the configuration transition is w_1 for the robot. Otherwise, if the bouncing angle changes, the weight of the configuration transition is w_2 for the robot.

In our approach, we modify the generalized cell-to-cell mapping method to find the roadmap $\mathcal{G}$ and the minimum navigation plan between x_I and x_G in the cell configuration space X_{free}.

To attain these, Algorithm 1 receives as input the geometric description of the environment E, a set of bouncing angles Φ, zero error range $\epsilon = 0$, the initial configuration x_I, and the goal configuration x_G. It produces the roadmap $\mathcal{G}$ and the observation string $\tilde{y}$, which is the minimal navigation plan as output.

In Algorithm 1, for each bouncing angle with zero error range $\phi_i \pm 0$ where $i \in \{1 \cdots |\Phi|\}$, from the set of bouncing angles Φ, each cell $z \in Z$ computes the configuration of the cell that represents the location (centroid) and the orientation of a cell (line 6). The next mapped cell z' represents the subsequent cell after z (line 7) which is calculated as:

$$\begin{aligned} x_{z'} &= x_z + \frac{r}{2}(u_l + u_r)\cos\theta, \\ y_{z'} &= y_z + \frac{r}{2}(u_l + u_r)\sin\theta, \\ \theta' &= \begin{cases} \theta, & \text{if } (x_{z'}, y_{z'}) \in E, \\ (\theta + \phi) \mod 2\pi, & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

where $(u_l, u_r) = (1, 1)$ specifies the left and right wheel velocities and $r = 1$ is the wheel radius of the robot.

If the new orientation of the cell θ' is equal to the previous orientation of the cell θ then the cell number of z' is calculated from the new cell center location and previous orientation, $(x_{z'}, y_{z'}, \theta)$, of z' (line 9). Cells z, z' are added to vertices set and their ordered pair (z, z') is added to edges set of the directed graph $\mathcal{G}_i$ and weight of the corresponding edge is updated with w_1 on $\mathcal{G}_i$ (lines 10–12). Otherwise, Algorithm 1 calculates the set of possible

bouncing cells with respect to the given bouncing angle set Φ. For the processing bouncing angle, we compute the next cell z' using the new cell center location and orientation of the cell with zero error range $(x_{z'}, y_{z'}, \theta' \pm 0)$ (line 16). Both cells, their ordered pair as an edge, and weight of their edge are added to $\mathcal{G}_i$ as before (lines 17–19). For all other bouncing angles that represent the change of bouncing angles, we compute the next cell z' again from the previous cell center location using Equation 3, the previous orientation, and the bouncing angle with zero error range $(x_z, y_z, \theta, \phi \pm 0)$ (line 21). Both cells and their ordered pair as an edge are added to $\mathcal{G}_i$ but the weight of their edge is updated with w_2 on $\mathcal{G}_i$ (lines 22–24).

Algorithm 1 ROADMAP&PLAN($E, \Phi, \epsilon, w_1, w_2, x_I, x_G$)

Input: $E, \Phi, \epsilon = 0, w_1, w_2, x_I, x_G$ {Environment, a set of bouncing angles, error range, weights, initial, and goal configuration}
Output: $\mathcal{G}, \tilde{y}$ {Roadmap, Observation String}

```
1: G ← ∅
2: for i = 1 to |Φ| do
3:    G_i.V ← ∅,   G_i.E ← ∅
4:    for j = 1 to N do
5:       z ← j
6:       x_z, y_z, θ ← CELLCONFIGUARTION(z)
7:       x_z', y_z', θ' ← NEXTCELL(x_z, y_z, θ, φ_i)
8:       if θ == θ' then
9:          z' ← CELLNUMBER(x_z', y_z', θ)
10:         G_i.V ← G_i.V ∪ {z, z'}
11:         G_i.E ← G_i.E ∪ {(z, z')}
12:         w(z, z') ← w_1
13:      else
14:         for k = 1 to |Φ| do
15:            if k == i then
16:               z' ← CELLNUMBER(x_z', y_z', θ' ± 0)
17:               G_i.V ← G_i.V ∪ {z, z'}
18:               G_i.E ← G_i.E ∪ {(z, z')}
19:               w(z, z') ← w_1
20:            else
21:               z' ←OTHERCELL(x_z, y_z, θ, φ_k ± 0)
22:               G_i.V ← G_i.V ∪ {z, z'}
23:               G_i.E ← G_i.E ∪ {(z, z')}
24:               w(z, z') ← w_2
25:            end if
26:         end for
27:      end if
28:   end for
29:   G ← G ∪ G_i
30: end for
31: τ ← SHORTESTPATH(G, x_I, x_G)
32: ỹ ← NAVIGATIONPLAN(τ)
33: return G, ỹ
```

We repeat the same process for all the bouncing angles in the given set Φ and create the roadmap $\mathcal{G}$ from the geometry. In line 31, we use Dijkstra's shortest path algorithm [22] to find the shortest path τ on the roadmap $\mathcal{G}$ from all ordered pairs of initial configuration x_I and goal configuration x_G for different bouncing angles in Φ. The function NAVIGATIONPLAN finally returns the observation string $\tilde{y}$ based on the shortest path τ (line 32). In this function, if the consecutive cell distance in the shortest path τ is less than N, it implies "forward movement" and gives '0' as one discrete observation. Otherwise, it implies the "bump" event and gives the bouncing angle ϕ as another discrete observation. This observation string $\tilde{y}$ provides the solution of single (x_I, x_G) navigation query.

B. Generating All Minimum Navigation Plans in Roadmap

We generate all minimum navigation plans from all possible shortest paths among all (x_I, x_G) pairs in the cell configuration space X_{free}.

To obtain all possible shortest paths, Algorithm 2 takes the roadmap $\mathcal{G}$ constructed from Algorithm 1 as input and generates all minimum navigation plans M from their shortest paths if one or more paths exist among ordered (x_I, x_G) pairs and their path weights L as output.

Algorithm 2 ALLPLANGENERATION($\mathcal{G}$)

Input: $\mathcal{G}$ {Roadmap}
Output: M, L {Navigation Plans, Path Weights}

```
1: let M[1..N, 1..N], L[1..N, 1..N] be 2D lists
2: for i = 1 to N do
3:    for i = 1 to N do
4:       M[i][j] ← NIL
5:       L[i][j] ← 0
6:    end for
7: end for
8: for i = 1 to N do
9:    for j = 1 to N do
10:      if i ≠ j then
11:         τ, l ← SHORTESTPATHANDWEIGHT(G, i, j)
12:         if τ ≠ NIL then
13:            L[i][j] ← −1
14:         else
15:            ỹ ← NAVIGATIONPLAN(τ)
16:            M[i][j] ← ỹ
17:            L[i][j] ← l
18:         end if
19:      end if
20:   end for
21: end for
22: return M, L
```

Algorithm 2 initializes M and L lists (lines 2–7). From $x_i \in X_{\text{free}}$ on the roadmap $\mathcal{G}$ to all other $x_j \in X_{\text{free}}$ on $\mathcal{G}$, we run Dijkstra's algorithm to find τ and minimum weight l among the (x_i, x_j) ordered pairs for the set of bouncing angles Φ on $\mathcal{G}$ (line 11). If τ is none, which means that there is no path among the (x_i, x_j) pairs, Algorithm 2 assigns -1 to the (i, j)-th entry of L (lines 12–13). Otherwise, it encodes the shortest path τ into an observation string $\tilde{y}$ as a minimum navigation plan (line 15) and then assigns the plan $\tilde{y}$ and minimum path weight l to the (i, j)-th entry of M and L respectively (lines 16–17). We find minimum navigation plans and path weights for all $x \in X_{\text{free}}$. Finally, Algorithm 2 returns M and L lists. These minimum navigation plans and path weights can then be used to answer multiple (x_I, x_G) navigation plan queries and their comparison.

Algorithm Analysis: The running time of the Algorithm 2 is $O(N^2 \log N)$ since it applies Dijkstra's shortest path algorithm to all (x_I, x_G) pairs for each $x \in X_{\text{free}}$.

C. Finding Bouncing Policy Distribution for Coverage

We combine all bouncing policies represented by the set of bouncing angles Φ for the given environment E to get the closest coverage to a target coverage distribution b over E. Let the probability of reliable rotation of the robot be r. We apply GCM method that uses the bouncing angle set Φ, the probability of reliable rotation r, and the nonzero error range $\pm\epsilon$. This method finds a number of persistent groups starting from all transient cells for each bouncing angle with an error range $\phi \pm \epsilon$. Since the persistent groups are the long-term behavior of the GCM, we consider the coverage

distribution of these persistent groups of a bouncing policy as the coverage of the environment by that bouncing policy. So, the transient cells are not considered for the coverage of the environment. A persistent group is an irreducible Markov chain as all its cells form a single communicating class. So, all persistent groups for a bouncing policy create a finite Markov chain $\mathcal{P}$. The limiting distribution π of $\mathcal{P}$ represents the coverage of E for each bouncing policy. First, we find the limiting distribution set Π for all the bouncing policies and then, using Π, we compute the bouncing policy distribution α of all bouncing policies through optimization.

In order to obtain the limiting distribution set Π, Algorithm 3 takes as input the geometric description of the environment E, the set of bouncing angles Φ, the error range ϵ, the probability of reliable rotation r. It returns Π as output.

Algorithm 3 PolicyDistribution(E, Φ, ϵ, r)

Input: E, Φ, ϵ, r {Environment, a set of bouncing angles, error range and probability of reliable rotation}
Output: $\Pi = \{\pi_1, \pi_2, \cdots, \pi_{|\Phi|}\}$ {A set of limiting distributions}

1: **for** $i = 1$ to $|\Phi|$ **do**
2: $G.V \leftarrow \emptyset, \quad G.E \leftarrow \emptyset$
3: **for** $j = 1$ to N **do**
4: $z \leftarrow j$
5: $x_z, y_z, \theta \leftarrow$ CELLCONFIGUARTION(z)
6: $x_{z'}, y_{z'}, \theta' \leftarrow$ NEXTCELL$(x_z, y_z, \theta, \phi_i)$
7: **if** $\theta == \theta'$ **then**
8: $z' \leftarrow$ CELLNUMBER$(x_{z'}, y_{z'}, \theta)$
9: $G.V \leftarrow G.V \cup \{z, z'\}$
10: $G.E \leftarrow G.E \cup \{(z, z')\}$
11: **else**
12: $Z' \leftarrow$ CELLSET$(x_{z'}, y_{z'}, \theta' \pm \epsilon)$
13: $G.V \leftarrow G.V \cup Z' \cup \{z\}$
14: $G.E \leftarrow G.E \cup \{(z, z'), z' \in Z'\}$
15: **end if**
16: **end for**
17: $S \leftarrow$ STRONGLYCONNECTEDCOMPONENT(G)
18: $T \leftarrow$ TRANSITIVECLOSURE(G)
19: $\mathcal{P} \leftarrow$ MCFROMPERSISTENTGROUP(S, T, r)
20: $\pi_i \leftarrow$ NORMALIZEDLIMITINGDISTRIBUTION$(\mathcal{P})$
21: $\Pi \leftarrow \Pi \cup \pi_i$
22: **end for**
23: **return** Π

In Algorithm 3, for each bouncing angle with error range $\phi \pm \epsilon$ from the set of bouncing angle Φ, we create a unweighted directed graph G following the same graph creation procedure of Algorithm 1 without adding weight to the edges of G. Additionally, for each cell $z \in Z$ when the new orientation of the cell θ' is not equal to the previous orientation of the cell θ, Algorithm 3 calculates the set of possible bouncing cells Z' where $Z' \subset Z$, using the new orientation of the cell with error range $\theta' \pm \epsilon$ (line 12). All cells z, Z' are added to the vertices set and their ordered pairs (z, z'), where $z' \in Z'$, are added the edges set of G for the processing bouncing angle (line 13-14).

Then, it finds the strongly connected component S from G using Tarjan's strongly connected component algorithm [23] (line 17). It also constructs the reachability matrix T, finding the transitive closure from G (line 18). From S and T, Algorithm 3 finds persistent groups using the function MCFROMPERSISTENTGROUP (line 19). In this function, if each vertex in a strongly connected component is reachable from all other vertices in the strongly connected component then this strongly connected component is found as a persistent group and each cell of this persistent group is classified as a persistent cell. If a vertex in a strongly connected component is reachable from a subset of vertices in the strongly connected component then each cell of this strongly connected component is classified as a transient cell.

Further, in MCFROMPERSISTENTGROUP function, an adjacency list is created from all persistent groups of the processing bouncing angle ϕ. Based on this adjacency list and reliable rotation probability r, the function creates the one-step transition probability matrix $\mathcal{P}$. To obtain this, the function uses the probability $p_{ij} = r$ for reliable rotation from cell z_i to cell z_j, $p_{ij} = \frac{(1-r)}{2\epsilon}$ for unreliable rotation from cell z_i to cell z_j, and $p_{ij} = 1$ for forward movement from cell z_i to cell z_j. In the last step, it calculates the limiting distribution π of $\mathcal{P}$ for the processing bouncing angle with the error range $\phi \pm \epsilon$, normalizes π, and adds it to Π (line 20-21). Finally, Algorithm 3 returns Π for all bouncing policies.

The probability distribution of choosing the bouncing policies for the robot can be represented as an m-dimensional vector where $m = |\Phi|$,

$$\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_m) \tag{4}$$

Equation 4 should satisfy: 1) $\alpha_i \geq 0$ for all $i \in \{1...m\}$, and 2) $\alpha_1 + \alpha_2 + ... + \alpha_m = 1$. The value α_i is the proportion of the robot choosing the i-th bouncing policy.

We obtain the optimal bouncing policy distribution α from the limiting distribution set Π to achieve as coverage as close as possible to the target coverage distribution b. We create the matrix A based on Π. We use the constrained least square [24] to compute the optimal bouncing policy distribution α which is given by the following optimization equation:

$$\begin{aligned} &\text{minimize} \quad \|A\alpha - b\|^2 \\ &\text{subject to} \quad C\alpha = d, \alpha >= 0 \end{aligned} \tag{5}$$

Here A is an $n \times m$ matrix, b is the n-vector where $n = N$, α is the m-vector, C is an $1 \times m$ matrix.

The bouncing policy distribution α is the optimal for obtaining the coverage closest to the target coverage distribution b because it minimizes the norm of the residual error $\|A\alpha - b\|$ having the constraints of Equation 5. This bouncing policy distribution α states the time-based switching among the bouncing policies to cover the environment.

V. EXPERIMENTAL RESULTS

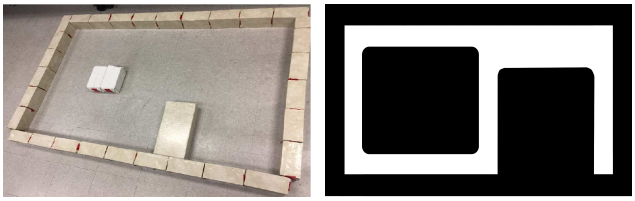
In this section, we present the simulation result and the physical implementation of our algorithms.

A. Minimal Navigation Plan Result

We have tested Algorithm 1 by developing a simulation and deploying it on a physical robot platform in the hardware experiment. We used the iRobot Create Roomba as a differential drive robot in an artificial laboratory environment of Fig. 2(a). The Roomba has many sensors but we utilized only the bump and clock sensors. In the simulation, the configuration space of the laboratory environment is computed analytically for the disk robot

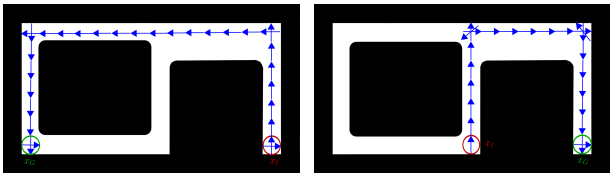
Roomba, as illustrated in Fig. 2(b). The cell configuration space X_{free} is discretized into $N = 384$ cells. In the simulation and experiment of navigation plans, we considered 8 different orientations of S^1 with $45°$ between each orientation.

We ran our simulation for the above discretized cell configuration space X_{free} using the set of bouncing angles $\Phi = \{45°, 90°, 135°\}$ and the error range $\epsilon = 0°$. We set the weight of using the same bouncing angle, $w_1 = 1$ and the weight of changing the bouncing angle, $w_2 = 100$ in our simulation. The illustrations of the first navigation plan between $x_I = 9$ and $x_G = 1$ and the second navigation plan between $x_I = 104$ and $x_G = 9$ are shown in Fig. 3. In Fig. 3(a), the first navigation plan of the robot uses only one bouncing angle, $90°$, to navigate from the bottom right corner of E, facing East, to the bottom left corner of E, facing East. In Fig. 3(b), the second navigation plan of the robot uses two bouncing angles, $90°$ and $135°$, to navigate from the bottom left corner of the obstacle that is touching ∂E, facing North to bottom right corner of E facing East. Our simulation gives two observation strings as output for two navigation plans; $\tilde{y}_1 = 00000090°00000000000000090°00000090°$ and $\tilde{y}_2 = 000000135°135°000000135°135°00000090°$.



(a) (b)

Fig. 2. A laboratory environment: (a) an environment using floor and bricks that includes one completely interior obstacle and one obstacle touching the boundary of the environment; (b) the configuration space of the environment shown in (a).

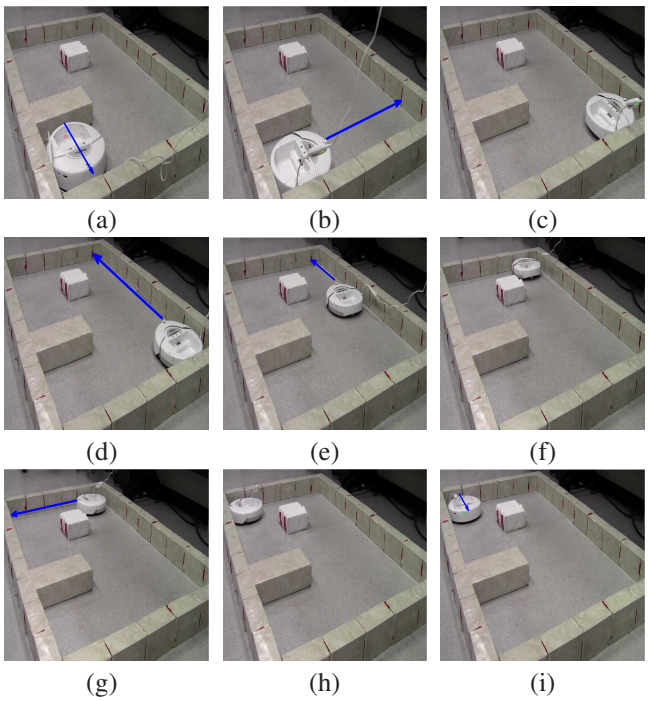


(a) (b)

Fig. 3. Simulation results of two navigation plans in the environment of Fig. 2: a) the path from the initial configuration (bottom right corner of E, facing East) to the goal configuration (bottom left corner of E, facing East). b) the path from the initial configuration (bottom left corner of the obstacle attached to ∂E, facing North) to the goal configuration (bottom right corner of E, facing East).

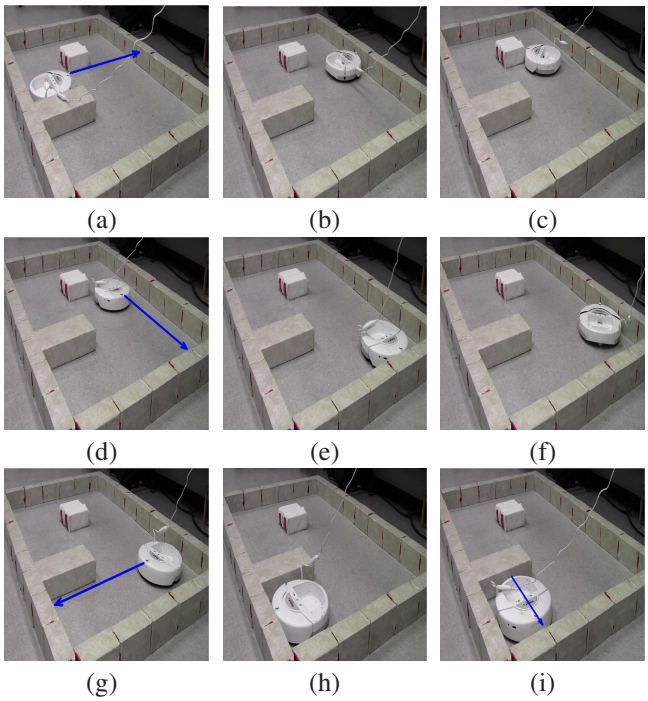
Afterward, we deployed the two generated observation strings $\tilde{y}_1$ and $\tilde{y}_2$ on the Roomba to navigate in the environment depicted in Fig. 2(a). We show snapshots of the two hardware experiments of the corresponding simulated navigation plans in Fig. 4 and 5. In our first hardware experiment of Fig. 4, we have placed the Roomba in $x_I = 9$ and it follows the observation string $\tilde{y}_1$ to get to $x_G = 1$. In our second hardware experiment (Fig. 5), we have also put the Roomba in $x_I = 104$ and it successfully reaches to $x_G = 9$. In these hardware experiments, the Roomba uses its clock to measure the number of zeros as it moves forward and the duration of rotation for different bouncing angles. It also uses bump sensors for detecting the "bump event".

To test Algorithm 1 in a more complex environment, the configuration space, as illustrated in Fig. 6, is discretized



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Fig. 4. Snapshots of different configurations of the robot executing the first navigation plan of the simulation result of Fig. 3(a): a) the initial configuration; $90°$ rotations are illustrated by the snapshot transitions a–b, c–d, f–g, and h–i; after snapshots b, d, e, and g, the robot moves forward; i) the goal configuration.

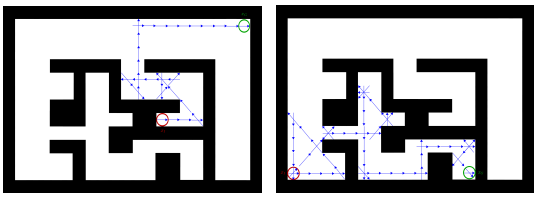


(a) (b) (c) (d) (e) (f) (g) (h) (i)

Fig. 5. Snapshots of different configurations of the robot executing the second navigation plan of the simulation result of Fig. 3(b): a) the initial configuration; $135°$ rotations are illustrated by the snapshot transitions b–c, c–d, e–f, f–g; a $90°$ rotation is illustrated by the snapshot transition h–i; after snapshots a, d, and g, the robot moves forward; i) the goal configuration.

into $N = 1464$ cells considering 8 different directions of S^1 that are $45°$ apart of each other. For two pairs of initial and goal configurations in the given cell configuration space, Algorithm 1 finds two navigation plans using the same set of bouncing angles $\Phi = \{45°, 90°, 135°\}$. In Fig. 6(a), the first navigation plan of the robot uses two bouncing angles, $90°$ and $135°$, to get to the goal configu-

ration x_G from the initial configuration x_I where locations of x_G and x_I are illustrated with 'red' and 'green' circles respectively. They face East-ward, and the navigation path is depicted with 'blue' arrows . In Fig. 6(b), the second navigation plan of the robot uses all bouncing angles, 45°,90°, and 135°, to complete its navigation task from x_I to x_G where the navigation path and its initial and goal configuration are illustrated the same way as before.



(a) (b)

Fig. 6. Simulation results of two navigation plans in a complex environment between pairs of initial configurations ('red' circle locations of the environment, facing East) and goal configurations ('green' circle locations of the environment, facing East).

B. All Minimum Navigation Plans Generation Result

We generate all feasible minimum navigation plans for all (x_I, x_G) pairs in the cell configuration space X_{free} of the environment depicted in Fig. 2 from the simulation of Algorithm 2. All minimum navigation plans and path weights on X_{free} using Φ are stored. We use all of these navigation plans to answer multiple (x_I, x_G) navigation plan queries. Then, we compare the total number of minimum navigation plans using different numbers of bouncing angles. The comparison result is shown in Fig. 7. This result suggests that most of the minimum navigation plans use only one bouncing angle, three bouncing angles are used more than two bouncing angles, and few of them use no bouncing angle, i.e., it does not bounce to get to the goal configuration.

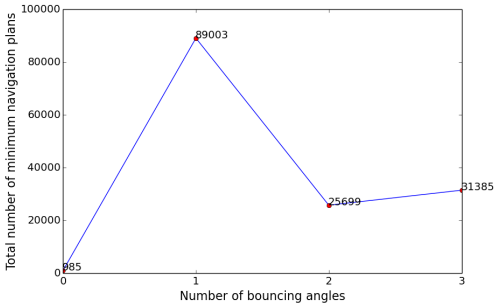


Fig. 7. Comparison of using each number of bouncing angles for generated minimum navigation plans in the environment depicted in Fig. 2.

We demonstrate one possible generated navigation plan from Algorithm 1 between the 'red' circled initial configuration x_I and the 'green' circled goal configuration x_G in the environment, as illustrated in Fig. 1. We also generate all feasible minimum navigation plans for all pairs (x_I, x_G) in X_{free} of the environment depicted in Fig. 1 from Algorithm 2. In this simulation, X_{free} has $N = 432$ cells considering the same 8 directions of S^1 with 45° between them and the set of bouncing angles $\Phi = \{45°, 90°, 135°\}$. Again, we store all minimum navigation plans and path weights, and compare the total number of navigation plans based on the number of bouncing angles. The comparison result is shown in Fig. 8. The result shows that most of its navigation plans use three bouncing angles. Fewer plans use two, one, or no bouncing angle. So, both comparison results show that the number of bouncing angles for all feasible navigation plans depends on the type of the environment or its complexity.

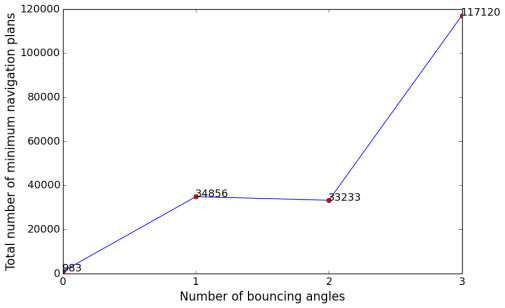
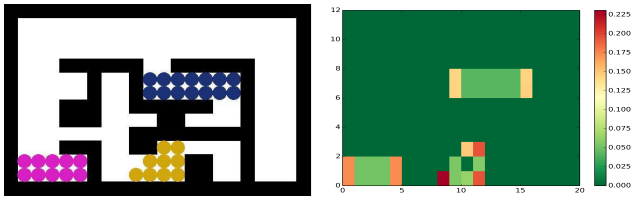


Fig. 8. Comparison of using each number of bouncing angles for generated minimum navigation plans in the environment depicted in Fig. 1.

In our simulation result, we do not have navigation paths for all pairs of (x_I, x_G) because for some of (x_I, x_G) pairs there is no path on the roadmap $\mathcal{G}$. However, the inclusion of more reliable bouncing angles in the set, Φ, will guarantee the navigation path for all pairs of (x_I, x_G).

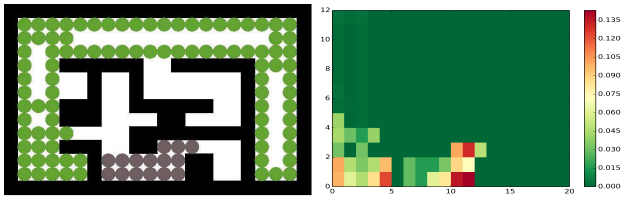
C. Result of Bouncing Policy Distribution for Coverage

We have used the same complex environment E of Fig. 6 for finding the bouncing policy distribution α to get as close to the uniform target coverage distribution b. The configuration space of the given environment X_{free} is discretized into $N = 65880$ cells considering $S^1 = [0, 2\pi)$. We ran the simulation of Algorithm 3 on X_{free} for the bouncing angle set $\Phi = \{30°, 75°, 315°\}$ as three bouncing policies, the rotation error range $\epsilon = \pm 5°$, and the probability of reliable rotation $r = 0.8$. Based on the output from Algorithm 3, the visualizations of persistent groups in E for different directions of S^1 and their corresponding heatmaps of the limiting distribution set Π for bouncing angles including error $30° \pm 5°$, $75° \pm 5°$, $315° \pm 5°$ are demonstrated in Fig. 9, 10, and 11 respectively. The heatmaps show the probability of visiting different locations in the environment over time by a robot starting from any location.



(a) (b)

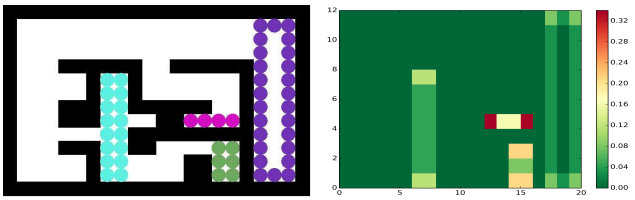
Fig. 9. Persistent groups visualization in E for different directions of S^1 and their corresponding heatmap of limiting distributions for the bouncing angle including error range, $30° \pm 5°$.



(a) (b)

Fig. 10. Persistent groups visualization in E for different directions of S^1 and their corresponding heatmap of limiting distributions for the bouncing angle including error range, $75° \pm 5°$.

We apply the constrained least square method of Equation 5 using the result of the limiting distribution set Π and



(a) (b)

Fig. 11. Persistent groups visualization in E for different directions of S^1 and their corresponding heatmap of limiting distributions for the bouncing angle including error range, $315° \pm 5°$.

the uniform target coverage distribution b. As a result, the optimal probability distribution α of all bouncing policies is computed, and is tabulated in Table I. This result implies that the robot has to use the bouncing policy of angle $315°$ about 50% of the time, and either the bouncing policy of angle $75°$ or the bouncing policy of angle $30°$ around 25% of the time.

TABLE I

OPTIMAL BOUNCING POLICY DISTRIBUTION RESULT

Bouncing policies, Φ	Proportion of choosing bouncing policies α
$30°$	0.22138172
$75°$	0.28005791
$315°$	0.49856037

VI. CONCLUSION AND FUTURE WORK

In this work, we have proposed navigation and coverage methods for a robot equipped with a clock and a contact sensor in a given environment. We constructed a directed graph from the environment using a set of bouncing policies. We found the minimum navigation plans on the graph between either one given pair of initial and goal configurations or all possible pairs of initial and goal configurations. The optimal bouncing policy distribution is calculated from the given set of bouncing policies to get the best possible coverage of the environment with respect to a target coverage distribution. We presented hardware experiments and simulation results of our method, which shows the usefulness of our work. However, there are still some future directions to extend our work.

We can include error in the rotation of the robot for navigation between the initial and goal configurations. Our navigation method can be extended for finding the navigation plan in an environment with dynamic obstacles. More reliable bouncing angles can be added to the set of bouncing angles for getting additionally minimal navigation plans. However, it is still an open problem to develop a complete navigation method for both rectilinear and non-rectilinear environments using the minimalist bouncing robot. Though we have done experiments in a lab environment but we should undertake these experiments in a more realistic office environment.

The coverage method can be augmented for a multi-robot version of our work. Each robot can cover different parts of interest in the environment without any coordination among these robots. This could be useful for developing distributed multi-robot patrolling methods in the environment too. Instead of finding the bouncing policy distribution, we can find particular number of best bouncing policies from all the bouncing policies represented by all bouncing angles in S^1. We could also develop a method to merge different bouncing policies into a single bouncing policy for a single robot to follow that policy in covering the environment effectively.

We plan to solve the *mapping* problem, constructing the map of the environment accessible by our robot using its simple bouncing behavior. We will develop a method for the *searching* problem utilizing the bouncing robot's movement around the environment until it gets to a specific spot in the environment.

ACKNOWLEDGMENTS

This work was supported in part by ARO grant 67736CSII. This work was also supported in part by NSF awards IIS-1302393, IIS-1527436, and IIS-1453652.

REFERENCES

[1] J. M. O'Kane and S. M. LaValle, "Localization with limited sensing," *IEEE Transactions on Robotics*, vol. 23, no. 4, p. 704, 2007.

[2] J. S. Lewis and J. M. O'Kane, "Guaranteed navigation with an unreliable blind robot," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 5519–5524, 2010.

[3] J. S. Lewis and J. M. O'Kane, "Reliable indoor navigation with an unreliable robot: Allowing temporary uncertainty for maximum mobility," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 160–165, 2012.

[4] J. S. Lewis and J. M. O'Kane, "Planning for provably reliable navigation using an unreliable, nearly sensorless robot," *The International Journal of Robotics Research*, 2013.

[5] B. Tovar, L. Guilamo, and S. M. LaValle, "Gap navigation trees: Minimal representation for visibility-based tasks," in *Algorithmic Foundations of Robotics VI*, pp. 425–440, Springer, 2004.

[6] T. Alam, M. Edwards, L. Bobadilla, and D. Shell, "Distributed multi-robot area patrolling in adversarial environments," in *International Workshop on Robotic Sensor Networks*, 2015.

[7] A. Lazanas and J. C. Latombe, "Landmark-based robot navigation," *Algorithmica*, vol. 13, no. 5, pp. 472–501, 1995.

[8] N. Roy, W. Burgard, D. Fox, and S. Thrun, "Coastal navigation-mobile robot navigation with uncertainty in dynamic environments," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 35–40, 1999.

[9] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *The International Journal of Robotics Research*, 2009.

[10] A. Agha-mohammadi, S. Chakravorty, and N. M. Amato, "On the probabilistic completeness of the sampling-based feedback motion planners in belief space," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 3983–3990, 2012.

[11] K. Hauser, "Randomized belief-space replanning in partially-observable continuous spaces," in *Algorithmic Foundations of Robotics IX*, pp. 193–209, Springer, 2010.

[12] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[13] S. C. Wong and B. A. MacDonald, "A topological coverage algorithm for mobile robots," in *Proc. of IEEE/RSJ International Conference on Intelligent Robot Systems*, pp. 1685–1690, 2003.

[14] E. Garcia and P. G. De Santos, "Mobile-robot navigation with complete coverage of unstructured environments," *Robotics and Autonomous Systems*, vol. 46, no. 4, pp. 195–204, 2004.

[15] C. K. Volos, I. M. Kyprianidis, and I. N. Stouboulos, "Experimental investigation on coverage performance of a chaotic autonomous mobile robot," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1314–1322, 2013.

[16] N. Agmon, G. A. Kaminka, and S. Kraus, "Multi-robot adversarial patrolling: facing a full-knowledge opponent," *Journal of Artificial Intelligence Research*, vol. 42, pp. 887–916, 2011.

[17] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[18] C. S. Hsu, "A theory of cell-to-cell mapping dynamical systems," *Journal of Applied Mechanics*, vol. 47, no. 4, pp. 931–939, 1980.

[19] L. Hong and J. Xu, "Crises and chaotic transients studied by the generalized cell mapping digraph method," *Physics Letters A*, vol. 262, no. 4, pp. 361–375, 1999.

[20] C. S. Hsu, *Cell-to-cell mapping: a method of global analysis for nonlinear systems*, vol. 64. Springer Science & Business Media, 2013.

[21] J. A. W. Van Der Spek, *Cell mapping methods: modifications and extensions*. PhD thesis, Eindhoven University of Technology, Netherlands, 1994.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.

[23] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[24] B. Gustafsson, "Least square problems," in *Fundamentals of Scientific Computing*, pp. 125–133, Springer, 2011.