Space-efficient filters for mobile robot localization from discrete limit cycles

Tauhidul Alam[1], Leonardo Bobadilla[1], Dylan A. Shell[2]

***Abstract*—Robot localization is the problem of determining a robot's pose in an environment, typically within a given map or a similar representation. Different methods have been proposed to address this localization problem for robots with limited sensing. In this paper, we present a localization method for a robot equipped with only a contact sensor and a clock. We make the limits of localization accuracy precise by establishing the fundamental limits imposed by symmetry as revealed by the robot's sensors. Our method is based on finding periodic cycles and transient trajectories of the robot path as it bounces within an environment filled with obstacles. Based on the cycles and transient trajectories, space-efficient and automata-based combinatorial filters are synthesized to solve localization problems modulo symmetries. Experimental results from multiple simulations and from real robot demonstrations attest to the feasibility and practicability of our method.**

***Index Terms*—Localization, mobile robots, cell-to-cell mapping, automata, sensor-based systems.**

I. Introduction

MOBILE robot localization is the problem of determining a robot's configuration (position and orientation) in its environment [1]. Localization is a fundamental problem in mobile robotics, and is typically a prerequisite to solving tasks such as navigation, coverage, mapping, searching, and patrolling for applications in agriculture, security, surveillance, and home robotics among many others. This work addresses the problem of global robot localization [2], [3], where a robot has to find its configuration in the entire environment without having any information about its initial configuration. Most localization approaches rely on *recursive Bayesian filters* such as particle [2], [4] or Kalman filters [5], [6], which, compared with the focus of our study, are far more expensive in terms of computation time and memory, and require sophisticated sensors and motion modeling. The originality of our work is that we synthesize a finite automata-based *combinatorial filter* for solving the problem of localizing a robot in a particular environment that is suitable for a device of meager computational ability, potentially even being realized directly in a field programmable gate-array. This work fits within a broader research program of hardware synthesis for robots.

Manuscript received: February 15, 2017; Revised: May 21, 2017; Accepted: July 6, 2017.

This paper was recommended for publication by Editor Kevin Lynch upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by ARO grant 67736CSII as well as in part by NSF awards IIS-1302393, IIS-1527436, and IIS-1453652.

[1]Tauhidul Alam and Leonardo Bobadilla are with the School of Computing and Information Sciences, Florida International University, Miami, FL, USA. {talam005, bobadilla}@cs.fiu.edu

[2]Dylan A. Shell is with the Department of Computer Science and Engineering at Texas A&M University, College Station, TX, USA. dshell@cse.tamu.edu

Digital Object Identifier (DOI): see top of this page.

With few or very limited sensors, the localization problem is challenging to solve and has consequently attracted considerable theoretical attention [7], [8], [9], [10]. The motivation of our work is to use a robot with limited linear and angular sensing as a basis for investigating the intrinsic limits of the localization problem. In particular, we wish to understand the strongest possible version of a localization task that such a robot can solve, recognizing that the robot may be too deficient, ultimately, to resolve its position down to a unique pose with certainty. What is possible depends on the environment and parameters of the robot controller, so we explore automated processes to uncover answers to these questions that are given in a particular setting as input.

Continuing the growing vein of work exploring the properties of sensing-constrained systems, we examine a robot equipped with a bump (or contact) sensor and a clock. The robot inhabits a planar polygonal environment with holes and has behavior parameterized by a single parameter, which, for reasons that will be obvious, we call the *bouncing angle*. The robot moves on straight lines, and when it encounters a wall it rotates through the bouncing angle (measured with respect to the direction of its pre-collision motion). While exhibiting the bouncing behavior, the robot's sensors provide a sequence of observations that will be specified in detail below.

Though the robot is too deficient to localize in the traditional metric sense, we show that there is a relaxed instance of the localization problem that it is capable of solving. The setting we study enables the construction of an estimator that still suffices to localize with an accuracy that is compromised only by the symmetries involved. This work is also motivated by the concepts of limit cycles and basins of attraction which we define here as periodic groups and transient trajectories respectively and are often used in control theory [11].

The main contributions of our work are as follows:

- We present an algorithm that is based on the *simple cell-to-cell mapping* [12] to find the periodic groups and their transient trajectories from the environment.
- We construct information state (I-state) graphs [1] from the computed periodic groups and transient trajectories.
- We introduce combinatorial filters that are generated from I-state graphs and enable the robot to localize itself up to some intrinsic uncertainty.

The remaining work is laid out as follows. Section II reviews the literature of robot localization and combinatorial filters. Section III defines the robot model, explains cell-to-cell mapping, and formulates problems we solve. In Section IV, we describe the method of our work in detail. Section V illustrates our simulation results and physical implementations of our work. Finally, we conclude with the discussion and outline the future work in Section VI.

II. Related Work

This section discusses related literature first on localization of mobile robots and secondly on combinatorial filters.

A. Robot Localization

There are several antecedent works which have examined bouncing robots in related contexts. In [13], [14], [8], the authors consider a robot whose bouncing angle varies as a function of the number of prior bounces. In [7], the bouncing angle of the robot is a constant angle relative to the normal of the impacted edge of the environment irrespective of its angle of incidence. These contrast from the type of bounce we study. The bounce we investigate ensures that the robot will end up in a small bounded set of possible locations.

In [9], [10], the authors study a robot equipped with a contact sensor and compass or a robot equipped with linear and angular odometers, providing theoretical results on localization for environments without holes using geometric reasoning. The robot they study is more powerful than what we explore herein. Further, holes within the environment pose no special challenge for the techniques we describe below. We note that in [8], the authors considered a simple environment with holes for localization with a robot having only a clock and a contact sensor. They presented a probabilistic technique for finding a probability distribution over regions on the boundary of the environment. However, they assume that the robot knows its initial orientation whereas in our work, no such assumption is needed.

B. Combinatorial Filters

Nearly sensorless robots called "weasel balls" that bump and bounce around the environment were studied in [15]. Their bounce is not associated with a fixed angle and thus require complete state estimation for solving different tasks. Since doing so is difficult, an information space view was introduced in [16] to avoid this onerous state estimation. The information space consists of all histories of actions and sensing observations of a mobile robot for problems involving uncertainty. A related perspective is adopted in the information state (I-state) formalism, which led to the use of combinatorial filters to process information from sensors for solving tasks such as manipulation [17], navigation [18], and target tracking [19]. In [20], the problem of filter reduction is introduced, which involves finding the filter that uses the fewest information states for a given filtering task. That paper showed that minimization of filters is an NP-complete problem; further, more recent work in [21], shows that several specially structured filters still retain this hardness property. And, though we do not claim that the filter which arises in our work is minimal in precisely the same sense as those authors, we note that the process we detail for constructing our localization filter only requires polynomial time and space. To the best of our knowledge, the present paper is the first to automate the process of compiling a geometric description of the environment into an I-state graph, which we then explicitly turn into a filter to solve the localization task.

III. Preliminaries

This section gives a description of the robot model we study. We also briefly explain the cell-to-cell mapping and formally define the localization problems of interest.

A. Robot Model

We start with a differential drive mobile robot equipped with only a contact sensor and a clock. The robot moves in a planar and bounded polygonal two-dimensional *workspace* $\mathcal{W} \subset \mathbb{R}^2$. There is a set of polygonal *obstacles* represented as $\mathcal{O} \subset \mathcal{W}$. Let $E = \mathcal{W} \setminus \mathcal{O}$ be free-space in which the robot can move freely and let ∂E represent the boundary of the free space. We assume that the robot has a map of the environment E and knows its bouncing angle ϕ but does not know its initial configuration. We also consider a noise-free model of the robot in terms of the translation and the rotation. Certainly, generating perfect motions for any angle poses a problem, especially for a low-cost differential drive robot. However, in practice, we found that for some given ϕ, we are able to produce repeatable and reliable rotations (see Section V, where we describe our physical robot experiments).

The robot moves straight until touching the boundary of the environment ∂E which is detected by the contact sensor. The robot measures the number of *steps* in its straight-line motion by using its clock. Once it bounces at ∂E, the robot rotates with the angle ϕ counterclockwise from its current orientation by commanding a constant angular velocity and using a clock to rotate for some fixed period of time. It then moves straight until contacting ∂E and repeats the behavior. This simple behavior is illustrated in Fig. 1.

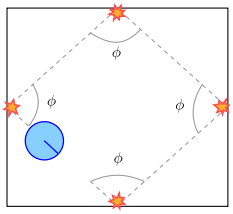

Fig. 1: An example of a simple bouncing robot.

B. Cell-to-Cell Mapping

Including the robot's orientation, the physical state space of the robot is $X = E \times S^1$ where $S^1 = [0, 2\pi)$. Let $x \in X$ denote the state of the robot where $x = (x_t, y_t, \theta)$, (x_t, y_t) is its position, and θ is its orientation. Let $R(x) \subset \mathbb{R}^2$ represent the robot. The obstacle region X_{obs} in the state space is defined as

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

and $X_{\text{free}} = X \setminus X_{\text{obs}}$.

The subset of the state space where the robot is allowed to move is denoted by X_{free}. To apply the cell-to-cell mapping method [12], [22], we divide X_{free} into equally sized 3-dimensional box cells since the robot's configuration has three degrees of freedom. Let N be the total number of cells. We define N as $N = N_E \times N_S$, where N_E is the discretization resolution of the 2-D free space E and N_S is the discretization resolution of S^1. This discretized state space is called *cell state space*. Each cell represents an indivisible state entity. The state of the system is described by a cell index $z \in \{1, \ldots, N\}$. Let $Z = \{1, \ldots, N\}$ denote the collection of cells.

The evolution of a system can be explained as a sequence of cells by investigating its state at discrete times. Let $e(i)$ denote the cell containing the state of the system at $t = i\Delta t$, $i = 0, 1, \ldots$ with Δt being the time between two state examinations, and being large enough to support crossing a cell. The system evolution is then governed by

$$e(i+1) = \mathcal{C}(e(i)) \tag{2}$$

where the mapping $\mathcal{C} : \mathbb{N} \to \mathbb{N}$ is called a simple cell-to-cell mapping (SCM). In this model, Eq.(2) implies that the next state of the system is determined entirely by its current state and is explicitly independent of the mapping step i.

For the sake of completeness, we summarize some important definitions of the cell-to-cell mapping method. An extended treatment can be found in [11].

Definition 1: (Periodic Cell) A cell z satisfying $z = \mathcal{C}^m(z)$, for some $m \in \mathbb{N}$ is called a periodic cell with a period of m.

Definition 2: (Transient Cell) A cell that is not periodic is called a transient cell and it maps into a periodic cell in a finite number of steps.

Definition 3: (Periodic Group) A sequence of K distinct cells $e(m)$, where $m = 1, 2, \ldots, K-1$, that satisfies

$$\begin{aligned} e(m+1) &= \mathcal{C}^m(e(1)), m = 1, 2, \ldots, K-1 \\ e(1) &= \mathcal{C}^K(e(1)), \end{aligned} \tag{3}$$

is called as a periodic group with a period K and each of the cells $e(\cdot)$ is said to be a periodic cell with the period K. This periodic group is also called an *attractor* or a *limit cycle*.

Definition 4: (Transient Trajectory) A transient trajectory is the set of initial cells that are finally leading to a particular periodic group (attractor). The collection of transient trajectories is called a *basin of attraction*.

C. Problem Formulation

As the robot moves in the environment E, it receives a sequence of *observations* from an *observation space* $Y = \{0, 1\}$. Given some agreed upon resolution, the robot can measure the distance by the number of *steps* between bounces up to some quantization error. For example, the observation string that represents the stream of observations that the robot in Fig. 1 is processing might be $\{0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0\}$, where the ones represent a bump event, and the zeros otherwise. Depending on scale, resolution, or both, there could be more than or less than three 0s between bumps. It is also possible to observe multiple 1s in a row (for example, it can happen when the robot bounces in a corner). This abstract symbolic representation can be realized with various implementations:

- The robot measures the number of steps for linear distance traversed since the last bump by a number of 0s. This measurement is quantized at some resolution.
- The robot moves forward at a constant speed and keeps observing a sequence of zeros. A bump event results in observing a 1 and resets the clock for the next linear distance measurement. These observations are encoded as a string of 0s, interspersed with 1s.

For a fixed bouncing angle ϕ, the cell-to-cell mapping method allows one to track the motion of the robot from any initial location in E and to find periodic groups, of which we assume there are r in total. Sometimes the robot's motion begins in a transient trajectory and sometimes it is already in a periodic group. It will eventually converge to one of the r periodic groups.

We are interested in the following problems:

Problem 1: Closed-world localization *Given an environment E, a bouncing angle ϕ, the fact that robot can only be within E, find the state of robot x as precisely as possible.*

Problem 2: Open-world localization *Given an environment E and a bouncing angle ϕ, find the state of robot x, determining whether the robot is within E and if so ascertain the state of the robot as precisely as possible; otherwise indicate that the robot is not in E.*

IV. Method

This section describes the sequence of steps that produce discrete filters for localization. It consists of the following steps: 1) find the periodic groups and transient trajectories and construct I-state graphs based on them; 2) create a nondeterministic automaton combining I-state graphs and 3) convert the nondeterministic automaton into a deterministic automaton to design filters that solve Closed- and Open-world localization problems.

A. Finding Periodic Groups & Transient Trajectories and Constructing Periodic Group I-State Graphs

In our method, we modify the simple cell-to-cell mapping to find all periodic groups (attractors) and associated transient trajectories (basins of attraction) [23] from a cell state space X_{free}. We also borrow the definition of an I-state graph from [20], though do away with the starting vertex.

Definition 5: (I-State Graph) An I-state graph $G = (V, E, \ell : E \to Y)$ is an edge-labeled directed graph where:

1) V is the finite set of vertices consisting of *I-states*.
2) E is the set of edges that represent transitions between vertices.
3) ℓ is the function that represents edges labeled by an observation in Y.

This I-state graph encodes the information state introduced by LaValle [1], integrating the history of observations made by a system during its execution. As the number of cells is finite, we can construct an I-state graph for each periodic group along with its transient trajectories, which we term a *periodic group I-state graph*.

In this step, Algorithm 1 receives as input the geometric description of the environment E and a bouncing angle ϕ, finds all r periodic groups P, consisting of periodic cells and transient trajectories T, consisting of transient cells, for these r periodic groups, and constructs a set of r periodic group I-state graphs denoted by $\mathcal{G}(V, E)$ as output.

In Algorithm 1, cells are assigned a group number and a step number. For each cell $z \in Z$, the group number g_z denotes the periodic group to which z belongs, the step number s_z denotes the number of mappings necessary for z to end up in a periodic group, and the next mapped cell is denoted by c_z. Initially, all cells are identified as *virgin cells* by assigning their group number zero. Each virgin cell $z \in Z$ determines the location (centroid) and orientation of a cell (line 6). In lines 7–12, a cell sequence z, $\mathcal{C}(z)$, $\mathcal{C}^2(z) \ldots \mathcal{C}^k(z)$ where $k \in \mathbb{N}$ and $k \leq N$, is generated for each virgin cell $z \in Z$ and cells in the sequence are identified as *cells under processing* by temporarily assigning to them their group number -1. The next mapped cell z' represents the subsequent cell after z. Thus, z' (line 9) is computed as:

$$\begin{aligned} x' &= x + \cos\theta, \\ y' &= y + \sin\theta, \\ \theta' &= \begin{cases} \theta, & \text{if } (x', y') \in E, \\ (\theta + \phi) \mod 2\pi, & \text{otherwise.} \end{cases} \end{aligned} \tag{4}$$

where ϕ is the bouncing angle of the robot. The cell number of z' is calculated from the center location and orientation, (x', y', θ'), of z' (line 10). Then, z' is stored in c_z and z is updated with z' (line 11). The generation of cell sequences is continued as long as z is a virgin cell, which also means it does not have the next mapped cell. This sequence generation is terminated in one of the following two cases:

1) If z has appeared again in the same sequence, thus forming a cycle. This case can be further subdivided into two scenarios, as illustrated in Fig. 2. In the first scenario of Fig. 2(left), when the initial and ending cells are the same, then all cells in the sequence are classified as periodic cells. In the second scenario of Fig. 2(right), when the initial and ending cells are different, then the cells prior to the cell that forms the cycle are classified as transient cells and the rest of cells, which form the entire cycle, are classified as periodic cells.
2) If z appeared in one of the previous sequences then all the cells in the sequence are classified as transient cells.

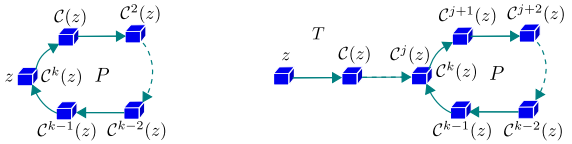

Fig. 2: Two cycle forming scenarios in the cell sequence: (left) Same initial and ending cells; (right) Different initial and ending cells.

Algorithm 1: MODIFIEDSIMPLECELLMAPPING(E, ϕ)

Input: E, ϕ – Environment and bouncing angle
Output: $\mathcal{G} = \{G_1, \ldots, G_r\}$ – Set of periodic group I-state graphs

1 $g[1..N] \leftarrow 0, \quad s[1..N] \leftarrow \infty, \quad c[1..N] \leftarrow \perp$
2 $r \leftarrow 0$
3 **for** $i \leftarrow 1$ *to* N **do**
4 **if** $g_i == 0$ **then**
5 $k \leftarrow 0, \quad z \leftarrow i$
6 $x, y, \theta \leftarrow$ CELLCONFIGURATION(z)
7 **while** $c_z == \perp$ **do**
8 $g_z \leftarrow -1$
9 $x', y', \theta' \leftarrow$ NEXTCELL(x, y, θ, ϕ)
10 $z' \leftarrow$ CELLNUMBER(x', y', θ')
11 $c_z \leftarrow z', \quad z \leftarrow z'$ // save and update next cell
12 $k \leftarrow k + 1$
13 **if** $g_z == -1$ **then** // new periodic group
14 $r \leftarrow r + 1, P_r \leftarrow \emptyset, T_r \leftarrow \emptyset$
15 **if** $i == z$ **then** // same initial and ending cells
16 **for** $j \leftarrow 0$ *to* $k - 1$ **do** // add periodic group
17 $g_i \leftarrow r, P_r \leftarrow P_r \cup \{i\}, s_i \leftarrow 0$
18 $i \leftarrow c_i$
19 **else** // different initial and ending cells
20 **for** $j \leftarrow 0$ *to* $d - 1$ **do** // cycle at d-th index
21 $g_i \leftarrow r, T_r \leftarrow T_r \cup \{i\}, s_i \leftarrow d - j$
22 $i \leftarrow c_i$ // add transient trajectory
23 **for** $j \leftarrow d$ *to* $k - 1$ **do** // add periodic group
24 $g_i \leftarrow r, P_r \leftarrow P_r \cup \{i\}, s_i \leftarrow 0$
25 $i \leftarrow c_i$
26 **else** // cell appeared in one of the previous sequences
27 **for** $j \leftarrow 0$ *to* $k - 1$ **do** // add transient trajectory
28 $g_i \leftarrow g_z, T_r \leftarrow T_r \cup \{i\}, s_i \leftarrow s_z + k - j$
29 $i \leftarrow c_i$
30 $\mathcal{G} \leftarrow \{$BUILDI-STATEGRAPH($P_i, T_i$) $\mid i \in \{1, \ldots, r\}\}$
31 **return** $\mathcal{G}$

All periodic cells in the j-th periodic group where $j \in \{1, \ldots, r\}$ are found with the update of their group number j and step number as zero (lines 16–18, 23–25). All transient cells in transient trajectories of the j-th periodic group are found with the update of their group number j and corresponding mapping number to get to the j-th periodic group as a step number (lines 20–22, 27–29).

For each periodic group and its associated transient trajectories, Algorithm 1 adds two consecutive cells to the vertex set, and their ordered pair to the edge set of the corresponding periodic group I-state graph, using the function BUILDI-STATEGRAPH (line 30). In this function, the absolute difference between orientations of these two consecutive cells, i.e., $|\theta - \theta'| > 0$ is checked. If the difference is not greater than zero, which means the robot moves forward with the same orientation, then the edge of this consecutive cell pair is labeled with 0. Otherwise, the transition between vertices causes a 'bump' event at the boundary of the environment $\partial E \subset E$ and the robot changes its orientation from θ to θ', thus the edge of this consecutive cell pair is labeled with 1. After construction, each periodic group I-state graph forms an octopus-like structure.

We repeat the same procedure for all r periodic groups and union the disjoint graphs. Thus, the set of r periodic group I-state graphs $\mathcal{G}$ is constructed. The total number of vertices of the r periodic group I-state graphs $\mathcal{G}$ is $|\mathcal{G}.V| = N$. We denote the set of vertices in the periodic groups of $\mathcal{G}$ as $\mathcal{G}.V_P$ where $\mathcal{G}.V_P \subset \mathcal{G}.V$. We illustrate one periodic group I-state graph in Fig. 3. In the periodic group I-state graph, vertices (cells) in a periodic group form a cycle and vertices (cell) in transient trajectory can terminate in one of two ways. It can terminate with its last vertex (cell) being either coincident with a cell (vertex) in the periodic group or coincident with a vertex (cell) in another transient trajectory which itself terminates in the aforesaid periodic group.

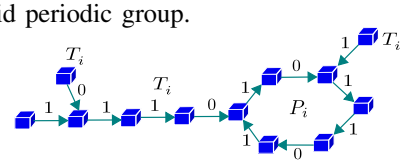

Fig. 3: A periodic group I-state graph.

Complexity of Algorithm: The running time of Algorithm 1 is $O(N)$ where N is the total number of cells since its complexity is dominated by line 4, which iterates over all the cells, processing each cell exactly once.

B. Creating Nondeterministic I-State Automaton

In the next step, we create a nondeterministic I-state automaton, A, amalgamating the entire set of periodic group I-state graphs $\mathcal{G}$ and define a nondeterministic I-state automaton as follows:

Definition 6: (Nondeterministic I-State Automaton) Let $A \triangleq (Q, \Sigma_\epsilon, \delta, q_0, F)$ be a nondeterministic automaton which accepts a stream of discrete observations from Y in which:

1) $Q = \{q_0\} \cup \mathcal{G}.V$ is a finite set of states.
2) $\Sigma_\epsilon = Y \cup \{\epsilon\}$ is a finite alphabet where $Y = \{0, 1\}$.
3) δ is the state transition function for any $q \in Q$ and any input alphabet $a \in \Sigma_\epsilon$ as below:

$$\delta(q, a) = \begin{cases} \{q'\} & q \in Q \setminus \{q_0\},\ a = \ell(q, q') \\ & \text{and } (q, q') \in \mathcal{G}.E, \\ Q \setminus \{q_0\} & q = q_0 \text{ and } a = \epsilon, \end{cases}$$

and $|\delta(q_j,0)|+|\delta(q_j,1)|=1, \forall j=1,\ldots,N$.
4) q_0 is the newly created initial state.
5) $F=\mathcal{G}.V_P$ is the set of final states that represents the set of vertices in periodic group I-state graphs.

The states of A except the initial state q_0 are essentially the same states (or vertices) as the periodic group I-state graphs $\mathcal{G}$. The number of states of A becomes $N+1$. A nondeterministic I-state automaton using only the periodic group I-state graph of Fig. 3 is illustrated in Fig. 4.

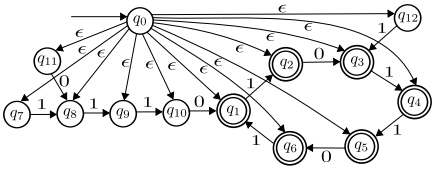

Fig. 4: A nondeterministic I-state automaton.

C. Nondeterministc I-State Automaton to Deterministic I-State Automaton Conversion

Given the nondeterministic I-state automaton A, we construct a deterministic I-State automaton A', converting the $\epsilon-$nondeterminstic automaton into a deterministic one using lazy evaluation method as follows:

Definition 7: (Deterministic I-State Automaton) Let $A' \triangleq (Q', Y, \delta', q_0', F')$ be a deterministic automaton that also accepts the stream of discrete observations from Y as [24] where:

1) $Q' = \{S : S \subseteq Q \text{ and } S = \epsilon\text{-Closure}(S)\}$ where ϵ-Closure(S) is the set that contains S including all states reachable from any state in S following one or more ϵ-transitions.
2) $Y = \{0, 1\}$.
3) $\delta'(S,a) = \bigcup \{\epsilon\text{-Closure}(p) : p \in \delta(s,a) \text{ for some } s \in S\}$.
4) $q_0' = \epsilon\text{-Closure}(q_0)$.
5) $F' = \{S : S \in Q' \text{ and } S \cap F \neq \emptyset\}$.

All transitions that are not defined lead to the 'trap' state implicitly. The converted deterministic I-state automaton A' produces a directed graph in which the outdegree of each state is at most two and each state represents one or more vertices of the periodic group I-state graphs $\mathcal{G}$. The states of A' that represent vertices in the transient trajectories of $\mathcal{G}$ form a directed acyclic graph. The states that represent the last vertices of transient trajectories lead to simple cycles (e.g. closed paths where no vertices and edges are repeated) in A'. We use the term *knowledge cycles* for these cycles. The states in the knowledge cycles of A' represent set of the vertices of periodic groups, $\mathcal{G}.V_P$, of $\mathcal{G}$. These knowledge cycles act like attractors; once the robot reaches one via states in the transient trajectories, it cannot leave.

Proposition 1: The number of states in the deterministic I-state automaton A' is $O(N^2)$ with respect to the number of states N in the nondeterministic I-state automaton A.

Proof: The only non-determinism in A is the ϵ-transitions from the initial state to all other states. Moreover, there are no transitions in A that return back to the initial state. Every transition, except the initial one, is deterministic as there is at most one observation (either a '1' or a '0') from a state. There are no self-loops in A because translation or rotation of the robot changes the underlying state of the system. There are two parts in A': the first part consists of states that represent set of states composed of both transient states and periodic states in A and the second part consists of states that represent set of periodic states in A. Let $N = N_t + N_p$ where N_t is the number of transients states in A, N_p is the number of periodic states in A. The states in the first part of A' form a full binary tree in the worst case because these states have two children, labeling two observations (0 and 1) on their transitions, and no child has more than one parent. In this part, the number of transient states decreases or remains same from the root to the leaves of the tree because applying the transition function δ' on the root q_0' that represents Q, for two observations creates two disjoint sets Q_1 and Q_2 such that $|Q_1|+|Q_2| \leq |Q|$ and subsequent states follow this inequality. Thus, it follows by induction that the height of the binary tree is $O(\log(N_t))$ and the total number of states in the first part of A' is $O(2N_t-1)$. This tree has at most N_t leaves that transition to knowledge cycles, which is the second part of A'. Hence, the second part of A' has at most N_t knowledge cycles; one cycle for each leaf. The length of each knowledge cycle is at most N_p because in the worst case the states in the cycle can include all periodic states N_p and each state represents one periodic state (or singleton) of N_p. Then, the total number of states in the second part of A' becomes $O(N_t N_p)$. Therefore, the number of states in A' is $O(N_t N_p) + O(2N_t - 1)$ or $O(N^2)$. ■

D. Filters for the Closed- and Open-World Problems

The final step produces filters to solve the two localization problems formulated in Section III. We follow the standard filter definition from [20]. The definitions of localization filters for closed and open world problems are as follows:

Definition 8: (Filter for the closed-world) A localization filter for the closed world problem is a tuple $\mathcal{F}_C \triangleq (Q', Y, \delta', q_0', c : Q' \to \mathbb{N})$, where function c augments the deterministic I-state automaton by adding a color to its states.

The filter $\mathcal{F}_C$ receives an observation string as input and reports a color as output. There are no final states in $\mathcal{F}_C$. Instead, we assign color 1 to every state that represents the transient trajectory vertices of $\mathcal{G}$. We assign the different color numbers to the states of different knowledge cycles ranging from 2 to one more than the number of cycles in A'. The same color number is assigned to every state of the same knowledge cycle. A filter for the closed world problem augmenting the deterministic I-state automaton is depicted in Fig. 5. Here, the states in two knowledge cycles are assigned green and cyan colors, and the states that are not in knowledge cycles are assigned the white color.

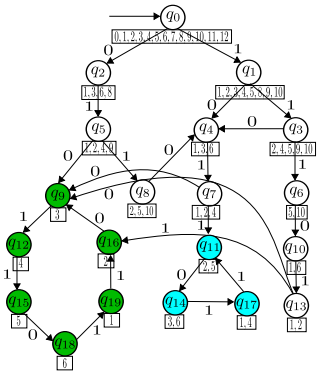

Fig. 5: A filter for the closed-world localization problem.

The filter $\mathcal{F}_C$ is used for localization of a robot in the closed-world problem case. When the robot enters into the colored knowledge cycle, it looks up the state $q' \in Q'$. Each state q' in the knowledge cycle of A' represents a set of states in the A. The cardinality of this set of states in A determines the uncertainty level of robot's position for solving localization problem. These states of A are also indexed by cell numbers. As these cell numbers indicate the configurations of the robot in E, the robot localizes itself. Depending on the aforementioned number, the robot may localize itself in one or more configurations in E. As an example, in Fig. 5, if the robot gets to a green knowledge cycle then it can localize up to a single configuration as it has an uncertainty of 1. On the other hand, if the robot gets to the cyan knowledge cycle then it can localize up to two configurations as it has an uncertainty of 2.

This filter solves the localization problem, and because it is a deterministic automaton, captures all the state needed to localize explicitly. The running time of F_C based on the deterministic I-state automaton is $O(n)$ where n is the length of an observation string as there is only one path through the automaton for the given observation string.

Definition 9: (Filter for the open-world)) A localization filter for the open world problem is a tuple $\mathcal{F}_O \triangleq (Q = Q' \cup \{q_t\}, Y, \delta', q_0', c : Q \to \mathbb{N})$. It augments the deterministic I-state automaton adding a "trap" state q_t along with assigning colors to all states.

The filter $\mathcal{F}_O$ also receives an observation string as input and reports a color as output to indicate whether the robot is in E or not. In the filter $\mathcal{F}_O$, there is no state transition for some states on a specific observation symbol. From these states on the missing observation symbol, we add transitions to the "trap" state q_t. We assign a new color number to q_t. Aside from this, we do the same process as $\mathcal{F}_C$ for the construction of $\mathcal{F}_O$. The "trap" state acts as a reject state in $\mathcal{F}_O$. Once the robot observes an observation string and if the evaluation of the observation string using $\mathcal{F}_O$ takes it to q_t, the robot can report that it is not in the environment E. Otherwise, $\mathcal{F}_O$ gives an output color as $\mathcal{F}_C$ which solves the closed world localization problem.

If the robot needs to localize itself in one of the k environments, then $\mathcal{F}_O$ can solve this problem too. For example, the robot knows a set $\mathcal{E}$ of three possible environments $\{E_1, E_2, E_3\}$ and some bouncing angle ϕ. Following the above method, we create three filters $\mathcal{F}_O$ for three environments. Then, we run them in parallel inside the robot. The robot will be able to declare that it is in one of these environments or not because of the "trap" state in the $\mathcal{F}_O$.

V. Experimental Results

In this section, we present an implementation of our proposed modified simple cell-to-cell mapping algorithm to find periodic groups and their transient trajectories for a point robot in a variety of environments and construct periodic group I-state graphs. We also provide experiments illustrating both simulation and physical implementation of the localization filters using an iRobot Create platform.

A. Simulation Results

We implemented the proposed modified simple cell mapping presented in Algorithm 1 in a simulation. The algorithm takes as an input the environment E and a bouncing angle ϕ and models the robot as a point.

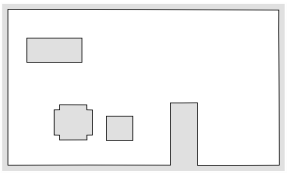

Fig. 6: A simple environment with three randomly placed obstacles (completely interior) and one static obstacle (touching boundary).

We set the size of the environment E of Fig. 6 to 200 $\times$ 125 grid unit lengths, excluding variable-size obstacles, $S^1 = [0, 2\pi)$. The cell size was set to 1 unit $\times$ 1 unit $\times$ 1°. We executed a simulation of Algorithm 1 and changing the obstacle region as follows:

- E_1: Randomly placing a square obstacle of fixed size inside the environment.
- E_2: Randomly placing a square and rectangular obstacles with fixed size inside the environment.
- E_3: Randomly placing a square, a rectangular, and a rectilinear obstacle with fixed size inside the environment.
- E_4: Randomly placing a scaled square obstacle inside the environment.

We ran the simulation of Algorithm 1 100 times for each of the four environments (E_1, E_2, E_3, E_4), keeping the bouncing angle $\phi = 90°$. We recorded the total number of periodic groups r and maximum transient trajectory length. Fig. 7left and right illustrate the values of r and maximum transient trajectories lengths. From these results, we conclude that values of r increase with the addition of obstacles and change with the scaling of an obstacle and also that the maximum transient trajectory length varies with increasing numbers of obstacles and the modification of the size of an obstacle. Some outliers are present in the plot of maximum transient trajectory length in Fig. 7(right) that are potentially useful for the coverage problem.

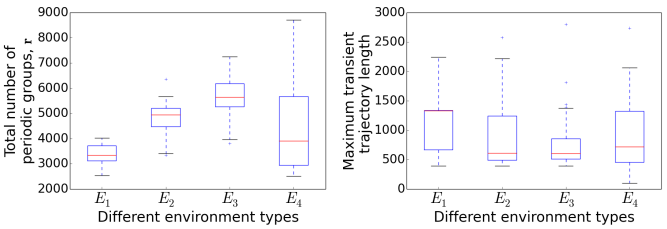

Fig. 7: A comparison of simulations for different environment types: (left) total number of the periodic groups r; (right) length of the longest transient trajectory.

B. From Simulation to Physical Implementation

We tested our Algorithm 1 with a differential drive robot, the iRobot Create/Roomba, in two environments using the bouncing angles $\phi = 45°, 135°$. The Roomba is equipped with many sensors but we only use the bump sensors and the clock. Since the Roomba is a disk robot, rather than a point robot, we analytically calculate the free configuration space X_{free} of the robot for both environments. For both environments, the free space which is also the cell state space X_{free} is discretized in $N = 152$ cells having 19 cells in each of 8 different orientations of S^1 with 45° separation between each orientation.

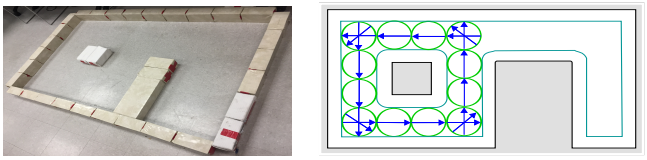

Fig. 8: The first lab environment (left) and the simulation result showing the visualization of the periodic group for this environment and the bouncing angle $\phi = 45^\circ$ (right).

We ran our first simulation test on the X_{free} of the environment of Fig. 8(left) using the bouncing angle $\phi = 45^\circ$ and our second simulation test on the X_{free} of the environment of Fig. 9(left) using the bouncing angle $\phi = 135^\circ$. We found $r = 1$ periodic group including its corresponding transient trajectories from the first simulation test and $r = 2$ periodic groups along with their corresponding transient trajectories from the second simulation test. We visualize one periodic group of our first simulation run in Fig. 8(right) and rest of the configurations are the transient trajectories part of the illustrated periodic group. We also show two periodic groups of our second simulation run in Fig. 9(right).

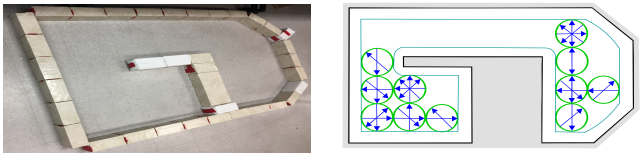

Fig. 9: The second lab environment (left) and the simulation result showing the visualization of all periodic groups for this environment and the bouncing angle $\phi = 135^\circ$ (right).

From the periodic group and the transient trajectories of the first simulation run, we constructed $\mathcal{G} = \{G_1\}$, the set of periodic group I-state graph. Based on $\mathcal{G}$, we created the nondeterministic I-state automaton A as shown in Fig. 10. In A, we added a new initial state and ϵ-transitions to all other states from it and made the states in the periodic group as final states. We made use of JFLAP [25] to create A. The indices of the states of A except the newly added initial state are the cell numbers in X_{free}.

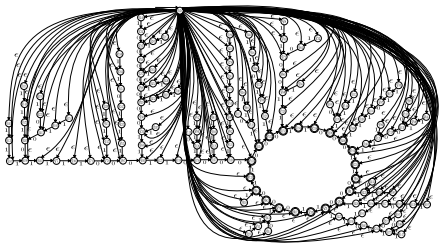

Fig. 10: Created non-deterministic I-state automaton for the environment and the simulation result of Fig. 8.

Again using JFLAP, we converted the nondeterministic automaton A into deterministic automaton A' as illustrated in Fig. 11. This deterministic automaton has 3 knowledge cycles. One of the knowledge cycles has an uncertainty of 1, one of them has an uncertainty of 3, and one has an uncertainty of 4. We created the localization filter for solving Problem 1, adding 3 colors to the states of the deterministic automaton. We colored the states outside of knowledge cycles white and chose 3 distinct colors for the states of 3 knowledge cycles. Next, we produced a filter for solving Problem 2 by adding a new "trap" state to the previous filter and we assign 5 colors to it as a new color is required for the "trap" state.

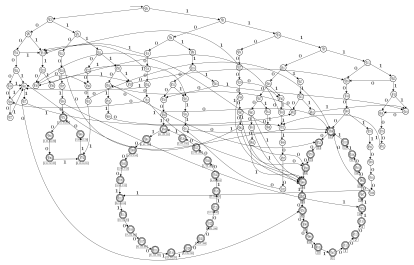

Fig. 11: Converted deterministic I-state automaton from the non-deterministic I-state automaton of Fig. 10.

We applied the same process to the periodic groups and the transient trajectories of the second simulation run and created the localization filters. We present the empirical results of 1) the number of states in deterministic I-state automaton A after converting from non-deterministic I-state automaton A', and 2) the computation time for this conversion in Table I. This conversion was performed on a GNU/Linux computer with Intel Core i7 3.6GHz processor and 16GB memory.

Table I: No. of states and computation time comparison.

Input		No. of states of non-deterministic I-state automaton, $N+1$	No. of states of deterministic I-state automaton	Computation time (sec.)
E	ϕ			
E of Fig. 8(left)	45°	153	129	24
	135°	153	124	20
E of Fig. 9(left)	45°	153	147	27
	135°	153	202	71

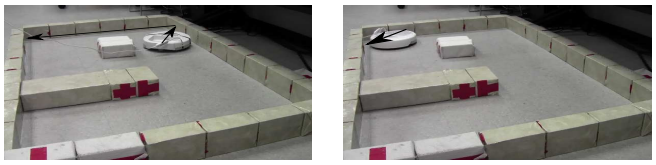

Fig. 12: Physical localization experiment in the environment of Fig. 8(left): (left) The robot was placed initially in the top left part of the environment; (right) after moving forward and bouncing with $\phi = 45^\circ$, it was localized up to 3 configurations in the periodic group visualized in Fig. 8(right).

Fig. 13: Physical localization experiment in the environment of Fig. 9(left): (left) The robot was placed initially in the top right part of the environment; (right) after moving forward and bouncing with $\phi = 135^\circ$, it was localized up to 1 configuration in one of the periodic groups visualized in Fig. 9(right).

We deployed the created localization filters on a Roomba and performed 10 physical experiments using the environments of Fig. 8 (left) and Fig. 9(left), and the bouncing angles $\phi = 45^\circ, 135^\circ$. Two of them are illustrated in Fig. 12 and Fig. 13. In these experiments, the robot was localized and stopped once a knowledge cycle of the filter was reached starting from the initial state. Since all states in each cycle represent the same cardinality of the set of configurations, the maximum and minimum number of configurations represented by the states where the robot was able to localize, are tabulated in Table II. Thus, the localization limits for an environment E and a bouncing angle ϕ are determined by the minimum

and maximum number of configurations of the robot, and the strongest possible localization is the minimum number of possible configurations of the robot.

Table II: Comparison of no. of localization configurations.

Input		Number of localization configurations	
E	ϕ	Minimum	Maximum
E of Fig. 8(left)	45°	1	4
	135°	1	2
E of Fig. 9(left)	45°	1	2
	135°	1	1

VI. Discussion and Future work

We have presented a localization method for a robot equipped with a contact sensor and a clock. Our method finds the periodic groups and their transient trajectories from a known environment and generates I-state graphs. We then use these I-state graphs to synthesize filters for solving localization problems. We demonstrated the practical feasibility of our localization solution in experiments with a real robot.

In practice, the online computation time of our localization filter is the time required to evaluate an observation string, which is linear with respect to the length of the observation string only. The offline construction of the filter is linear in terms of the number of cells, and the conversion from the nondeterministic automaton to the deterministic automaton is quadratic in terms of the number of cells. We adapted the comparison of different mobile robot localization methods from [26] (see Section 7.8) by adding our combinatorial filter (CF) based localization method as illustrated in Table III to show the pros and cons of the proposed method. These localization methods use stronger robot sensing models with cameras and range sensors which make them more robust compared to our sensing model, having only the clock and contact sensors. In addition, our cell-to-cell mapping based algorithm is scalable for a large configuration space by decomposing the whole free space into subspaces and processing them independently which allows its use in large environments. Several interesting directions remain for future work as described below.

Table III: Comparison of different localization methods.

	EKF	MHT	Coarse (topological) grid	Fine (metric) grid	MCL	CF (our method)
Measurements	Landmarks	Landmarks	Landmarks	Raw measure-ments	Raw measure-ments	Time and bump measure-ments
Measurement noise	Gaussian	Gaussian	Any	Any	Any	None
Posterior (on new observation)	Gaussian	Mixture of Gaussian	Histograms	Histograms	Particles	Single state
Efficiency (memory)	++	++	+	-	+	$O(\log N^2)$
Efficiency (time)	++	+	+	-	+	+++ [1]
Ease of implementation	+	-	+	-	++	++ [2]
Resolution	++	++	-	+	+	+ [3]
Robustness	-	+	+	++	++	- [4]
Global localization	No	No	Yes	Yes	Yes	Yes

We wish to reduce the uncertainty of the robot's configuration in the environment analyzing the structure of our localization filter. In order to do this, the robot may have to switch its bouncing angle and we can find the minimum number of changes to reduce the uncertainty. Since we discretize the state space and use a finite abstraction, our ideas connect naturally with *finite bisimulations* [27], [28]. We assume that our cell-to-cell abstraction is deterministic and also that the system resets to the cell center in each step. Since we can bound the execution time by the longest path from the initial state to a knowledge cycle, the time horizon of the localization is bounded and our assumptions may hold for short time intervals. Alternatively, we can model the nondeterministic cell-to-cell transitions by using *Generalized Cell-to-Cell Mapping (GCM)* [11] which uses a probabilistic cell transition map. We have implemented and used GCM to model the imperfect rotation of the robot for solving the coverage problem [29] and we believe that it can be used to account for the non-determinism in our localization method.

[1]Our filter takes constant time for the sensor update on a new observation.
[2]Ease of implementation of our filter is the same as MCL.
[3]The resolution of our method is similar to the fine (metric) grid method.
[4]Our method is not robust to the noise or erroneous output.

References

[1] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.
[2] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.
[3] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *J. of Artificial Intelligence Research*, vol. 11, pp. 391–427, 1999.
[4] F. Dieter, "Adapting the sample size in particle filters through kld-sampling," *The Int. J. of Robotics Research*, vol. 22, no. 12, pp. 985–1003, 2003.
[5] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, 1991.
[6] L. Jetto, S. Longhi, and G. Venturini, "Development and experimental validation of an adaptive extended kalman filter for the localization of mobile robots," *IEEE Trans. on Robotics and Automation*, vol. 15, no. 2, pp. 219–229, 1999.
[7] L. H. Erickson and S. M. LaValle, "Toward the design and analysis of blind, bouncing robots," in *Proc. of IEEE ICRA*, pp. 3233–3238, 2013.
[8] L. H. Erickson, J. Knuth, J. M. O'Kane, and S. M. LaValle, "Probabilistic localization with a blind robot," in *Proc. of IEEE ICRA*, pp. 1821–1827, 2008.
[9] J. M. O'Kane and S. M. LaValle, "Almost-sensorless localization," in *Proc. of ICRA*, pp. 3764–3769, 2005.
[10] J. M. O'Kane and S. M. LaValle, "Localization with limited sensing," *IEEE Trans. on Robotics*, vol. 23, no. 4, pp. 704–716, 2007.
[11] C. S. Hsu, *Cell-to-cell mapping: a method of global analysis for nonlinear systems*, vol. 64. Springer Science & Business Media, 2013.
[12] C. S. Hsu, "A theory of cell-to-cell mapping dynamical systems," *J. of Applied Mechanics*, vol. 47, no. 4, pp. 931–939, 1980.
[13] 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 ICRA*, pp. 160–165, 2012.
[14] J. S. Lewis and J. M. O'Kane, "Planning for provably reliable navigation using an unreliable, nearly sensorless robot," *The Int. J. of Robotics Research*, vol. 32, no. 11, pp. 1342–1357, 2013.
[15] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. M. LaValle, "Controlling wild bodies using linear temporal logic," in *Proc. of RSS*, 2012.
[16] B. Tovar, A. Yershova, J. M. O'Kane, and S. M. LaValle, "Information spaces for mobile robots," in *Proc. of RoMoCo*, pp. 11–20, 2005.
[17] S. M. Kristek and D. A. Shell, "Orienting deformable polygonal parts without sensors," in *Proc. of IEEE/RSJ IROS*, pp. 973–979, 2012.
[18] B. Tovar, R. Murrieta-Cid, and S. M. LaValle, "Distance-optimal navigation in an unknown environment without sensing distances," *IEEE Trans. on Robotics*, vol. 23, no. 3, pp. 506–518, 2007.
[19] J. Yu and S. M. LaValle, "Shadow information spaces: Combinatorial filters for tracking targets," *IEEE Trans. on Robotics*, vol. 28, no. 2, pp. 440–456, 2012.
[20] J. M. O'Kane and D. A. Shell, "Automatic reduction of combinatorial filters," in *Proc. of IEEE ICRA*, pp. 4082–4089, 2013.
[21] F. Z. Saberifar, M. R. Ali Mohades, and J. M. O'Kane, "Combinatorial Filter Reduction: Special Cases, Approximation, and Fixed-Parameter Tractability," *J. of Computer and System Sciences*, vol. 85, pp. 74–92, May 2017.
[22] D. H. V. Van Campen, E. L. B. Van De Vorst, J. A. W. Van Der Spek, and A. De Kraker, "Dynamics of a multi-dof beam system with discontinuous support," *Nonlinear Dynamics*, vol. 8, no. 4, pp. 453–466, 1995.
[23] J. A. W. Van Der Spek, *Cell mapping methods: modifications and extensions*. PhD thesis, Eindhoven University of Technology, Netherlands, 1994.
[24] J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*. Addison Wesley, MA, first edition, 1979.
[25] JFLAP. Available at http://www.jflap.org/.
[26] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.
[27] A. J. Van der Schaft, "Equivalence of dynamical systems by bisimulation," *IEEE Trans. on Automatic Control*, vol. 49, no. 12, pp. 2160–2172, 2004.
[28] P. Tabuada and G. J. Pappas, "Finite bisimulations of controllable linear systems," in *Proc. of IEEE CDC*, vol. 1, pp. 634–639, 2003.
[29] T. Alam, L. Bobadilla, and D. A. Shell, "Minimalist robot navigation and coverage using a dynamical system approach," in *Proc. of IEEE IRC*, pp. 249–256, 2017.