2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Sensor Coverage Robot Swarms Using Local Sensing without Metric Information

Rattanachai Ramaithitima, Michael Whitzer, Subhrajit Bhattacharya and Vijay Kumar *

Abstract— **We consider the problem of deploying a swarm of mobile robots into an unknown environment for attaining complete sensor coverage of the environment. The robots have limited and noisy sensing capabilities and no metric or global information available to them. Using tools from algebraic topology, we formally describe the sensor coverage as a simplicial complex, deploy robots through the complex using bearing-based local controllers, and attain coverage while identifying and removing sensor redundancies. Despite the highly limited sensing capabilities and complete lack of global localization and metric information, we demonstrate that the proposed algorithm is complete, always terminates in a finite-sized environment, is guaranteed to attain complete coverage and is robust to sensor failures. The algorithm presented in this paper was demonstrated through simulation and proves to effectively cover and explore unknown indoor environments.**

I. INTRODUCTION

A. Motivation and Related Work

Sensor coverage of indoor environments using teams of mobile robots is a well-studied problem in robotics. *Coverage path planning* refers to the task of visiting every point (or within a certain distance of every point) in a given environment as has been addressed in [1], [2], [3]. In the present context however, we focus on attaining *coverage by a sensor network*, which is the task of deployment and distribution of a team of robots such that they attain and maintain constant sensory coverage of every point in the environment. In presence of limited number of robots this problem has often been handled using Voronoi partition of the environment and minimization of a coverage functional [4], [5]. However such approaches invariably rely on a global and centralized localization capability for each robot (for example, using GPS). Complete sensor coverage of indoor environments using swarm of robots have been studied in [6], [7], where the known world is modeled as a graph, robots are assumed to have global localization and can be made to navigate independently from one location to another in a global coordinate frame. In almost all these lines of research, global localization of the robots, a priori knowledge of the environment (obstacle configurations), availability of metric information and ability to control the robots from one point in the environment to another have been assumed.

Biologically inspired multi-robot coverage algorithms have also been proposed [8], [1], which are most often distributed and the robots rely on local sensing only. Similar local communication-based algorithms for robot swarms have been used to construct various shapes [9] in an environment. However, such behavior-based algorithms come with very limited theoretical guarantee. Likewise, distributed coverage algorithm with no global localization have been studied in [10]. But the notion of coverage being purely based on a graph gives limited to no guarantee on the attainment of sensor coverage or the optimality. Furthermore such approaches inherently assume availability of some metric information.

In recent years coverage by sensor network has been studied more formally using *simplicial complex* and *homological tools* from algebraic topology [11], [12]. Such approaches are completely topological and require little to no metric information. In general, homology computation has been extremely useful in detecting holes in sensor network coverage. While some progress has indeed been made in controlling the network so as to mend the holes in sensor coverage [13], [14], all these methods work only in obstacle-free environments and require some localization of the robots for being able to control them.

Closely related to our work is the literature of exploration of unknown environment without global localization. Simultaneous Localization and Mapping (SLAM) [15] requires a robot (or a group of robots) to navigate in an environment acquiring range measurements (to obstacles), and then *stitch* the collected data to construct a complete map of the environment. This process is however quite complex and requires significant amount of computation. We assume limited computation power on each robot, and having a swarm of roots at our disposal and sensor coverage of the environment being the primary objective, we do not perform a full-blown SLAM. The robots' sensor capabilities are also limited compared to what is typically required by SLAM.

A similar research conducted by Lee, *et al.* [16] studies the problem of maximization of coverage area with limited number of robots equipped with limited local sensors. However, our approach is different, and performs better, in at least three aspects: First, our proposed method is robust to robot failure and can start from an arbitrary configuration of the robots, since the computations at every iteration are purely based on the current state only. Second, our approach of "pushing" the robots through the graph in order to expand the frontier to the unknown regions, instead of navigate a robot from the source to the frontier, has a lower execution time at each cycle, and does not require any restriction on the minimum distance between two neighboring robots. Lastly, our proposed algorithm does not require the workspace to be

*University of Pennsylvania. [ramar,mwhitzer,subhrabh, kumar]@seas.upenn.edu. The authors gratefully acknowledge the support of AFOSR grant FA9550-10-1-0567, ONR grants N00014-07-1-0829, N00014-09-1-1051, and N00014-09-1-103.

simply-connected, as we demonstrate through experimental results in complex, indoor environments with obstacles.

We adopt a simplicial complex representation (in particular, a Rips complex) as the richer and more formal description of the sensor coverage, without using metric information, to solve the problem of deployment of robots in an unknown environment without global localization. The robots only have onboard sensors to measure bearings to neighboring robots in their local coordinates and touch sensors for collision avoidance. Such limited sensing data essentially allows deployment of robots in a dark room, with the robots using their camera to identify their neighbors by illuminated (LED) markers on each other. Since our approach is fundamentally topological, the proposed method is highly robust to sensor noise. We attain complete sensor coverage of the entire finite environment assuming sufficient number of available robots, along with guarantees on coverage, exploration and a notion of local optimality.

B. Problem Description

We consider the problem of efficiently exploring an unknown indoor environment with a rapidly expanding swarm of robots with limited and noisy local sensing with no global localization or sensing capabilities. In particular, the only sensory capabilities that we assume on each robot are those of an omni-directional camera with a limited radial range of vision and a touch sensor to detect contact/collision with obstacles and other robots. We call the disk around a robot, representing a sensing radius of the omni-directional camera, the robot's *disk of visibility*, within which the bearing to the neighboring robots and their identities can be detected. However, the camera does not provide a range measurement due to projection of the spacial world on to the camera plane. Thus, obstacles are detected through touch sensors near the base of the robots, and cannot be detected using the camera. The robots can also communicate with each other and with a central router through wireless communication. The robots are assumed to holonomic (*i.e.* can be driven in arbitrary direction). We assume that there are sufficiently large number of robots available, which are being deployed from one or multiple sources (*e.g.* entrance to an environment).

The contribution of this paper is to design a distributable control algorithm for the robots such that they deploy themselves to attain hole-less coverage of the entire environment using their disks of visibility. That means, at least one robot should be able to *see* each and every point in the environment after the coverage is attained. However, the only information that they have are the bearing to their neighbors in their respective local coordinate frames and a directional touch information with obstacles. This means that there is *absolutely no metric information available*.

II. Preliminaries

A. Notations

We denote by $W \subset \mathbb{R}^2$ the obstacle-free region where the robots are being deployed and the sensor coverage of which needs to be attained. If there are n robots deployed in W, we assign IDs to them, $1, 2, \cdots, n$, and represent their joint configuration by $X = [x_1, x_2, \cdots, x_n],\ x_i \in W$.

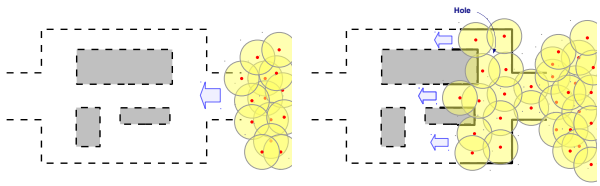
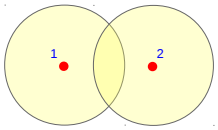


Fig. 1. Illustration of a swarm of robots entering an environment and attaining coverage. The hole shown on the right figure is something we would like to avoid.

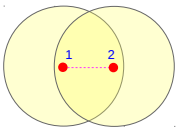
A robot, i, can measure the bearing to a neighbor, $j \in N_i$, in its local coordinates, where $N_i = \{j \mid \|x_i - x_j\| \leq r_v\}$ are the neighbors of i. We call this measurement $\theta^i_j \in [-\pi, \pi)$. If it measures the bearing to another robot, k as θ^i_k, then we define $\theta^i_{jk} = \big((\theta^i_k - \theta^i_j) \bmod 2\pi\big) - \pi$ — *i.e.*, the bearing to k relative to the bearing to j (and the angle converted to a value in $[-\pi, \pi)$).

B. Vietoris–Rips Complex of Camera Sensing Footprints

We do not assume that the robots can localize themselves and the only way of sensing/identifying neighbors is by using the camera. Thus, if the disks of visibility of two robots merely overlap, there is no way of detecting that fact (Figure 2(a)). We need to use a *stronger notion of overlap* — two robots know that their disks of visibility overlap if and only if they are in each other's disks of visibility and their line of sight is not blocked (Figure 2(b)).



(a) Robots can't see each other, and hence have no way of detecting that their disks of visibility overlap.



(b) Robots can see each other, and hence know that their disks of visibility overlap. Visibility is represented by the dotted magenta line.

Fig. 2. Detection of overlap of *disks of visibility* using only local visibility-based sensing.

We can thus consider a simplicial complex [17] representation of the free space that the camera footprints cover. The description of the abstract simplicial complex goes as follows (Figure 3): We add one 0-simplex to the complex for every deployed robot in the environment (the 0-simplices are identified by the robot IDs). A 1-simplex is added between two 0-simplices if the corresponding robots are in each other's disk of visibility (*i.e.*, are within distance of r_v from each other) and can see each other. A 2-simplex is added to the complex for every 3-tuple of robots that can all see each other (and hence their disks of visibility has a non-empty intersection). We do not construct any 3 or higher dimensional simplices since on a planar environment with obstacles, we are only concerned with the H_1 homology.

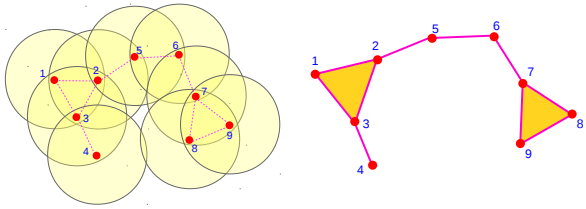


Fig. 3. Nine robots, their disks of visibility and the corresponding abstract simplicial complex, $\mathcal{R}_{r_v}$.

This simplicial complex is, by definition, the Vietoris-Rips complex [11] (or simply the *Rips complex*) on the set of robots (constructed up to dimension 2 – *i.e.*, we do not construct the 3 and higher simplices), with distance between pairs of robots being the length of the unobstructed line segment connecting them (and a distance being infinity if such a line segment does not exist due to presence of obstacles) and the parameter for the Rips complex being r_v. For a particular joint configuration of the robots, X, we call this simplicial complex $\mathcal{R}_{r_v}(X)$. We denote the 0-simplices in $\mathcal{R}_{r_v}(X)$ by the corresponding robot ID (*e.g.*, 'i'), the 1-simplices by pairs of IDs of the robots that make them up (*e.g.*, $\{i, j\}$), and 2-simplices by three tuples (such as $\{i, j, k\}$). Formally, $\mathcal{R}_{r_v}(X)$ is a chain complex [17] and constitutes of a sequence of modules (or vector spaces) along with boundary maps: $0 \xrightarrow{0} \mathcal{C}_2 \xrightarrow{\partial_2} \mathcal{C}_1 \xrightarrow{\partial_1} \mathcal{C}_0 \xrightarrow{0} 0$, where $\mathcal{C}_d$ is an abstract module (or vector space) generated (or spanned) by the d-dimensional simplices. Whenever the robots' configuration, X, is obvious or implied by the context, we will write $\mathcal{R}_{r_v}$ to denote the corresponding complex for simplicity. In Section III-A we will identify two sub-complexes of $\mathcal{R}_{r_v}$, namely the frontier subcomplex, $\mathcal{F}$, and obstacle subcomplex, $\mathcal{O}$, which together constitute the *fence subcomplex* [11], $\mathcal{K} = \mathcal{F} \cup \mathcal{O}$.

It is important to note that $\mathcal{R}_{r_v}$ can be constructed with local visibility information only, as described earlier, and does not require the entire configuration, X, of the robots to be known in a centralized manner. Furthermore, as will be evident in the next section, we do not need to construct the entire simplicial complex in a centralized fashion for most of the algorithmic components. It's only when we compute generators for the relative homology $H_2(\mathcal{R}_{r_v}, \mathcal{K})$, for optimization purposes, that we will need to store $\mathcal{R}_{r_v}$ in a centralized manner.

C. Contact/Touch Sensing Model

As mentioned earlier, the only sensors on board each robot are the omnidirectional camera and a collection of touch/contact sensors at the base of the robots. The camera is used to measure bearing with other robots inside the disk of visibility and is incapable of measuring range. The touch sensors are binary sensors, and are triggered when in contact with an obstacle/wall or another robot (Figure 4). The presence of multiple touch sensors (N_T counts of them) at the base also provides a rough estimate of direction of contact (within an error of $\tau = \frac{\pi}{N_T}$ when a single touch sensor is activated).

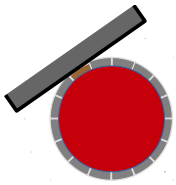


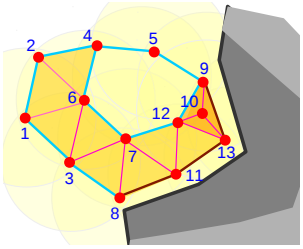
Fig. 4. The touch/contact sensors (gray protrusions) at the base of a robot (red). Contact with an obstacle or another robot triggers one or more touch sensors providing a rough estimate of the direction of contact.

D. Local Bearing-Based Controller

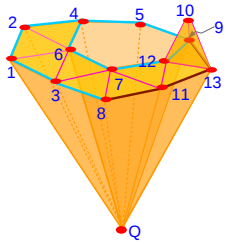
The robots are controlled using the bearing-based *visual homing controller* presented in [18]. This controller utilizes a gradient decent approach where desired bearing angles to landmarks are used to drive robots. The distances between a robot and its respective landmarks are not known. The only information known are the bearing angles, as is consistent with the assumptions in our paper. With the proper selection of the cost functional for the optimization process, the gradient of the path from start to goal (the velocity control command for robot i) is given by $v^i = K \sum_{j \in \mathcal{L}^i} (\theta^i_{j,\text{des}} - \theta^i_j)$, where, $\mathcal{L}^i$ is the list of robots that are neighbors of i, and which can be used as landmarks, $\theta^i_{j,\text{des}}$ is the desired bearings with landmark j, and K is a gain. Note that this velocity can be computed in the instantaneous local coordinate frame of the robot i. This controller converges to the goal configuration when the number of landmarks is greater than or equal to two and not co-linear with the goal location. In our implementation, the controller incorporates adaptive gain scaling in order to obtain faster convergence. We also use adaptive landmark detection depending on a robot's neighbor list while moving.

E. Relative H_2 Homology

We periodically compute a non-trivial 2-cycle of the relative complex $(\mathcal{R}_{r_v}, \mathcal{K})$ so that we can identify redundant/extra robots that can be removed from the complex without sacrificing sensor coverage. This is a direct application of the result in [11]. We assume some familiarity with algebraic topology and homology theory for the discussion in this section [17]. Given the simplicial complex, $\mathcal{R}_{r_v}$, and the fence subcomplex, $\mathcal{K} = \mathcal{F} \cup \mathcal{O}$, one can construct a relative chain complex, $C_*(\mathcal{R}_{r_v}, \mathcal{K})$. This, in essence, is the complex obtained by quotienting out the subcomplex $\mathcal{K}$ (*i.e.*, collapsing $\mathcal{K}$ to a single point, or introducing a new 0-simplex, Q, and connecting 2-simplices $\{i, j, Q\}$ to every $\{i, j\} \in \mathcal{K}$ – as illustrated in Figure 5). Due to a result from [11], if we can find a non-trivia *relative cycle* in $C_2(\mathcal{R}_{r_v}, \mathcal{K})$ such that it passes through all the 0-simplices in the fence, $\mathcal{K}$, then the 0-simplices (the robots) making up that cycle is sufficient for maintaining the sensor coverage. All other robots can be reallocated.



(a) A Rips complex, $\mathcal{R}_{r_v}$, with the frontier subcomplex, $\mathcal{F}$, marked in cyan, and the obstacle subcomplex, $\mathcal{O}$, marked in brown.



(b) Topology of the space where the entire fence complex, $\mathcal{F} \cup \mathcal{O}$, have been connected through 2-simplices to an external 0-simplex, namely Q.

Fig. 5. An example where a redundant robot (#10) can be identified from a non-trivial cycle in the space constructed by connecting all the fence simplices to a single external 0-simplex.

III. Algorithm Design

The outline of our swarm coverage algorithm is presented in Algorithm 1. We begin by deploying robot 1 into the unknown environment using an open-loop control so that it maintains visual contact with the source/base. Then, in line 3,

we start our deployment cycle by constructing the Rips complex in a distributed manner as described in Section II-B (and Algorithm 2). Using the current Rips complex $\mathcal{R}_{r_v}$, we update the frontier, $\mathcal{F}$, and obstacle, $\mathcal{O}$, subcomplexes (line 4, and Algorithm 3) along with computing the target location for deployment of the new robot in the local coordinates of a frontier robot. Note that at the end of the first deployment, the robot 1 belongs to $\mathcal{F}$ (as described in Section III-A). Although our implementation uses a centralized server that collects local information from the individual robots through an emulated wireless communication channel, most of the algorithmic components described in this section can be done in a decentralized fashion.

We periodically compute relative H_2 homology to identify redundant robots for redeployment (line 5). We then find the shortest path to a frontier robot (from the source or a redundant robot) through the 1-skeleton of the complex (line 6), along which we execute the "push" action (described in Section III-B) for deployment of the next robot (line 7). In presence of multiple sources, deployment can be performed in parallel along multiple paths as long as the paths do not intersect (which can be computed using an optimal routing algorithm).

Algorithm 1 Swarm Coverage Overview

1: Deploy Robot 1; $n \leftarrow 1$
2: **do**
3: Construct Rips cplx. through local communication: $\mathcal{R}_{r_v}$ = ComputeRipsComplex($\{N_i\}_{i=1,2,\cdots,n}$)
4: Compute fence subcplx. using local bearing info.: $[\mathcal{F}, \mathcal{O}]$ = FenceSubcomplex($\mathcal{R}_{r_v}, \{\theta^a_{bc}\}$)
5: (Periodically) Identify redundant robots using H_2 hom.
6: Find *path* in 1-skeleton for "Pushing" robots
7: "Push" robots in *path*
8: Deploy $(n+1)^{th}$ robot; $n \leftarrow n+1$
9: **while** $\mathcal{F} \neq \emptyset$

Algorithm 2 $\mathcal{R}_{r_v}$ = ComputeRipsComplex($\{N_i\}_{i=1,2,\cdots,n}$)
Input: Neighbor information, $N_i,\ i = 1, 2, \cdots, n$.
Output: Rips complex, $\mathcal{R}_{r_v}$.

1: $\mathcal{R}_{r_v} \leftarrow \emptyset$
2: **for** Robot $i = 1, ...n$ **do**
3: $\mathcal{R}_{r_v} \leftarrow \mathcal{R}_{r_v} \cup \{i\}$
4: **for** Robot $j \in N_i$ **do**
5: $\mathcal{R}_{r_v} \leftarrow \mathcal{R}_{r_v} \cup \{i, j\}$
6: **for** Robot $k \in N_i$ **do**
7: **if** $j \neq k$ and $j \in N_k$ **then**
8: $\mathcal{R}_{r_v} \leftarrow \mathcal{R}_{r_v} \cup \{i, j, k\}$
9: **end if**
10: **end for**
11: **end for**
12: **end for**

A. Identifying Frontier and Obstacle Subcomplexes

At a particular robot configuration, X, we identify the 1-simplices (and the corresponding 0-simplices that constitute) in the Rips complex, $\mathcal{R}_{r_v}$, which form the *frontier* to the unexplored regions, as well as the ones that are adjacent to the obstacles. They respectively constitute the frontier subcomplex, $\mathcal{F}$, and the obstacle subcomplex, $\mathcal{O}$. We define the *fence subcomplex* as $\mathcal{K} = \mathcal{F} \cup \mathcal{O}$.

Algorithm 3 $[\mathcal{F}, \mathcal{O}]$ = FenceSubcomplex($\mathcal{R}_{r_v}, \{\theta^a_{bc}\}$)
Input: Rips cplx., $\mathcal{R}_{r_v}$; Relative bearings, $\theta^a_{bc}, \{a, b, c\} \in \mathcal{R}_{r_v}$.
Output: Frontier subcomplex, $\mathcal{F}$; Obstacle subcomplex, $\mathcal{O}$

1: $\mathcal{F} \leftarrow \emptyset$, $\mathcal{O} \leftarrow \emptyset$
2: $E \leftarrow$ ComputeException($\mathcal{R}_{r_v}$)
3: **for** $\{i, j\} \in \mathcal{R}_{r_v} \setminus E$ **do**
4: UnCov$_{ij} \leftarrow \{+1, -1\} \setminus \{\text{sign}(\theta^i_{jk_u}) | \{i, j, k_u\} \in \mathcal{R}_{r_v}\}$
5: **if** UnCov$_{ij} \neq \emptyset$ **then** // A side of $\{i, j\}$ is uncovered.
6: $[\theta^i_{j,new}, \theta^j_{i,new}] \leftarrow$ DeploymentAngle(i, j, UnCov$_{ij}$)
7: **if** $\{i, j\}$ is an obstacle simplex **then**
8: $\mathcal{O} \leftarrow \mathcal{O} \cup \{\{i\}, \{j\}, \{i, j\}\}$
9: **else if** $\{i, j\}$ is a fronter simplex **then**
10: $\mathcal{F} \leftarrow \mathcal{F} \cup \{\{i\}, \{j\}, \{i, j\}\}$
11: **end if**
12: **end if**
13: **end for**

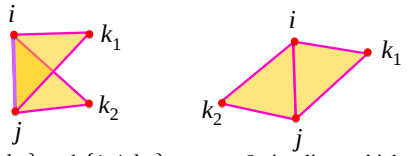
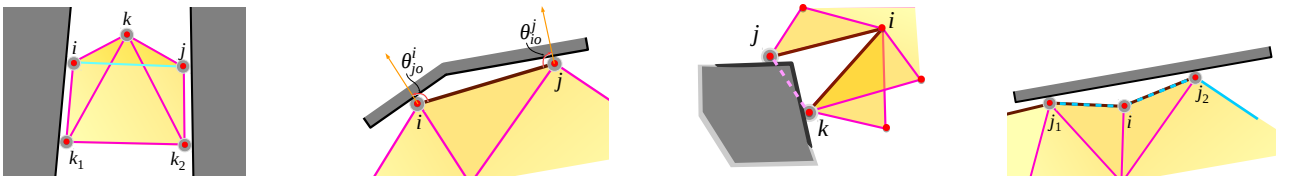


Fig. 7. $\{i, j, k_1\}$ and $\{i, j, k_2\}$ are two 2-simplices which have $\{i, j\}$ in their boundaries. $\{i, j\}$ is a fence 1-simplex if both k_1 and k_2 lie on the same side of $\overline{ij}$ (thick purple line in left figure), otherwise not (right figure).

Algorithm 3 describes our method of identifying the frontier subcomplex and obstacle subcomplex. We begin (line 2) by identifying the 1-simplices in $\mathcal{R}_{r_v}$ that are part of an exception set, E, as described later in Section III-A.1 (Figure 6(a)). For each 1-simplex $\{i, j\}$ in $\mathcal{R}_{r_v}$ that is not in E, we then compute the sign of $\theta^i_{jk_u}$ for all of the 2-simpilces $\{i, j, k_u\} \in \mathcal{R}_{r_v}$ (*i.e.*, the ones which have both i and j as their vertices). If all the 2-simplices adjacent to $\{i, j\}$ lie on the same side of the 1-simplex (Figure 7), then the bearing angle to all the robots k_u relative to j (resp. i) have the same sign, and thus in line 4, UnCov$_{ij}$ is not empty. Thus, i, j belongs to the fence subcomplex, and we compute and store the location (in the local coordinates of i and j) for potentially deploying a new robot to expand the frontier using a "pushing" action (line 6). The exact computation of the bearings to the potential new location, $\theta^i_{j,new}, \theta^j_{i,new}$, is described in subsection III-A.2 and Algorithm 4.

Finally, in lines 7-11, we classify each fence simplex, $\{i, j\}$, as *frontier* or *obstacle* using touch sensor readings and the outputs of DeploymentAngle as follows:

i. If i and j are in contact with an obstacle (*i.e.*, a touch sensor is activated, and there are no robots visible in the direction of the activated touch sensor) in the expanding direction, then the 1-simplex $\{i, j\}$ and 0-simplices i, j are placed in $\mathcal{O}$ (Figure 6(b)).
ii. Otherwise, we check for possibility of $\{i, j\}$ being an obstacle simplex at a convex corner as follows: We compute the "closest" other fence 1-simplex attached to i and j (this is computed as a part of the DeploymentAngle procedure – say it is $\{i, k\}$). If the magnitude of the angle between $\overline{ik}$ and $\overline{ij}$ is less than $\frac{\pi}{3} - 2\beta$ (figure 6(c), where β is the error in measurement of bearings to neighbors), then the two robots, j and k, do not see each other due to occlusion by an obstacle, but every free point (points outside



(a) Exception case where a 1-simplex, $\{i, j\}$, has all adjacent 1-simplices lying on the same side, but is not a fence simplex. This can be detected from the perspective of robot k.

(b) Detecting that a 1-simplex, $\{i, j\}$, is in $\mathcal{O} \subseteq \mathcal{R}_{r_v}$ (thick brown line).

(c) Convex corner case where a pair of 1-simplices, $\{i, j_1\}$ and $\{i, j_2\}$, are recognized as obstacle 1-simplices (thick brown lines).

(d) If the robot i is to be "pushed" along a path in the graph to expand frontier $\{i, j_1\}$, it performs a "test drive" to ensure an obstacle is not right in front of it.

Fig. 6. Identifying simplices for fence subcomplex $\mathcal{K} = \mathcal{F} \cup \mathcal{O}$.

obstacles) in their convex hull is in the visibility disk of at least one robot (aside from possibly small non-convex sub-features present in that convex corner, which we ignore). Thus, these 1-simplices are marked as obstacle 1-simplices to be pushed into $\mathcal{O}$.

iii. Otherwise, at least one of the robots can be expanded/moved to the unexplored region, and thus $\{i, j\}$ is placed in $\mathcal{F}$ along with the corresponding robots (Figure 7, left).

iv. Additionally, if $\{i, j\} \in \mathcal{F}$ due to 'iii.', and i belongs to the *path* for planned deployment, we perform a "test drive" in the planned deployment direction for a small distance to ensure sufficient space availability for new deployment near obstacles (Figure 6(d)).

The complete illustration of the process of identifying 1-simplices as part of $\mathcal{F}$ or $\mathcal{O}$ is given in Figures 7 and 9(a). We next describe the COMPUTEEXCEPTION and DEPLOYMENTANGLE procedures.

1) The Exception Case: The aforesaid approach in detecting fence 1-simplices using UnCov_{ij} may give false positives in some cases when a 1-simplex, $\{i, j\}$, is completely covered by 2 simplices, of which $\{i, j\}$ do not form a boundary, as shown in Figure 6(a). Nevertheless, this special case can be easily detected from the perspective of a common neighbor, k, of i, j. If it is detected that $\theta^k_{ij} = \theta^k_{ik_1} + \theta^k_{k_1k_2} + \cdots + \theta^k_{k_rj}$ (for some $k_1, \cdots, k_r \in N_k$), such that all the summands have the same sign as the summation, then clearly $\{i, j\}$ lies inside 2-simplices of which $\{i, j\}$ do not form a boundary but k is a vertex. Then $\{i, j\}$ is marked as an *exception* 1-*simplex.*

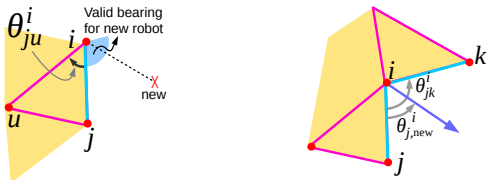
2) Identifying Locations for Robot Placement (Hexagonal Packing): Given a 1-simplex $\{i, j\} \in \mathcal{F}$ and the uncovered direction $\sigma \in \{+1, -1\}$, we need to find, in the local coordinates of i and j, the location for the new robot position. Figure 8(a) illustrates the uncovered side of 1-simplex $\{i, j\}$ in i's local coordinate. Our strategy for choosing the position to deploy next robot is to try and achieve a *hexagonal packing* [19] (which is the most optimal packing on an obstacle-free pane) of robots as much as possible, only to be interrupted by the presence of obstacles or control's error. This essentially boils down to sending robots at an angle of $60°(= \frac{\pi}{3})$ with respect to $\overline{ij}$ into the free region. Algorithm 4 describes our DEPLOYMENTANGLE function which first determines (lines 3-6) the "closest" other fence 1-simplices attached to i and j (*e.g.*, $\{i, k\}$ in Figure 8(b)). If there is no other fence 1-simplex attached to i, we set

Algorithm 4 $[\theta^i_{j,new}, \theta^j_{i,new}]$ = DEPLOYMENTANGLE $(i, j, \text{UnCov}_{ij})$

Input: Robots i, j; the *side* of $\overline{ij}$ that is open/uncovered.
Output: New location for deployment in local coordinates of i, j, or, $\{i, j\}$ is marked an obstacle simplex.

1: $\theta^i_{j,new} \leftarrow \emptyset$, $\theta^j_{i,new} \leftarrow \emptyset$
2: **for** σ in UnCov_{ij} **do**
3: $\quad S_i \leftarrow \{l \mid \{i, l\} \in \mathcal{R}_{r_v}$ and $\text{sign}(\theta^i_{j,l}) = \sigma\}$
4: $\quad k_i \leftarrow \arg\min_{k' \in S_i} |\theta^i_{j,k'}|$
5: $\quad S_j \leftarrow \{l \mid \{j, l\} \in \mathcal{R}_{r_v}$ and $\text{sign}(\theta^j_{i,l}) = -\sigma\}$
6: $\quad k_j \leftarrow \arg\min_{k' \in S_j} |\theta^j_{i,k'}|$
7: $\quad$ **if** $|\theta^i_{j,k_i}| < \frac{\pi}{3}$ (or $|\theta^j_{i,k_j}| < \frac{\pi}{3}$) **then**
8: $\quad\quad$ Mark $\{i, k_i\}$ (or $\{j, k_j\}$) as an obstacle simplex.
9: $\quad$ **else**
10: $\quad\quad \theta^i_{j,new} \leftarrow \sigma \min\{\frac{\pi}{3}, |\frac{\theta^i_{j,k_i}}{2}|\}$
11: $\quad\quad \theta^j_{i,new} \leftarrow -\sigma \min\{\frac{\pi}{3}, |\frac{\theta^j_{i,k_j}}{2}|\}$
12: $\quad$ **end if**
13: **end for**

$\theta^i_{\text{new}} = \theta^i_j + \sigma^i_j \frac{\pi}{3}$ — the 60° angle for deployment in a hexagonal packing. Otherwise we set the angle to the minimum between the one for hexagonal packaging ($\frac{\pi}{3}$) and the the one that bisects $\theta^i_{jk_i}$. Likewise for θ^j_{new}.



(a) The free side of $\{i, j\}$ where $\text{sign}(\theta^i_{j,new}) = \sigma$.

(b) The bearing to the new location, $\theta^i_{j,\text{new}} = \min\{\frac{\pi}{3}, \frac{\theta^i_{jk}}{2}\}$, in i's local coord.

Fig. 8. Determining bearing to the new location.

If i is not attached to a frontier 1-simplex (*e.g.*, i is a frontier robot in a narrow passage with a single file of robots), then we simply choose the direction *away* from the neighbors of i as the bearing to the new location (in the local coordinates of i) for deployment of the new robot.

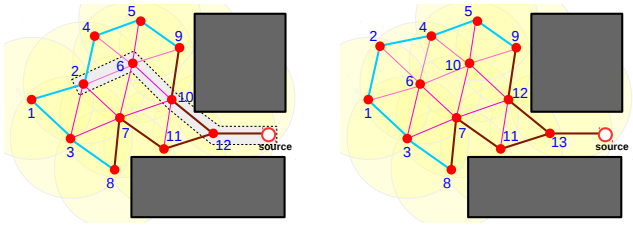
B. Identifying Path in 1-skeleton for "Pushing" Robots

The strategy in our algorithm for robot deployment in every control cycle is to keep the structure of the existing simplicial complex (and hence the positions of the existing robots in W) unchanged. New robots are deployed through the complex simply by *"pushing"* through paths (*i.e.*, making each robot on a path move forward to take the place of the one in front of it) in the 1-skeleton (graph) of the complex (Figure 9). For computing this path, a centralized knowledge

of the entire 1-skeleton is used (constructed by the robots communicating each of their local information – the IDs of the neighbors that each see – to a central server via wireless communication), although the computation of the path can indeed be performed in a decentralized manner through peer-to-peer communication only (see [20] for a decentralized implementation of the Dijkstra's algorithm).

We consider the graph made out of the 1 and 0 simplices in $\mathcal{R}_{r_v}$. The frontier subcomplex, $\mathcal{F}$, computed in previous section (the 0-simplices in it) provides the list of robots which we need to potentially move to *expand* the frontier. We assign a cost of 1 to all the 1-simplices in the graph, except the 1-simplies in $\mathcal{O}$, to which we assign cost of $w_O > 1$ in order to avoid paths that pass through robots adjacent to obstacles, where navigation is more challenging. We use Dijkstra's search algorithm to find the shortest path from the source, which is the robot next to the base station (in case of multiple source, we can initiate the *open list* in Dijkstra's algorithm with the multiple sources as illustrated in [5]), to the closest vertex (0-simplex) in $\mathcal{F}$.

Robots are then "pushed" along this path where each robot on the path simply gets replaced by the one behind it on the path, while the robot that is on the frontier computes (as described next) and moves to a new location in the free/unexplored region. Since robots on the path get *replaced* by the robots behind them, this requires that we not only update the IDs in $\mathcal{R}_{r_v}$, but also the robot IDs in $\mathcal{F}$ and $\mathcal{O}$.



(a) Shortest path $12 \to 10 \to 6 \to 2$ identified from the source to a vertex in $\mathcal{F}$.

(b) Robots are "pushed" along the path. Notice how the new robot 13 appears near the source.

Fig. 9. The complex $\mathcal{R}_{r_v}$, and the subcomplexes $\mathcal{F}$ (cyan) and $\mathcal{O}$ (brown). Path through the 1-skeleton illustrate *"pushing"*

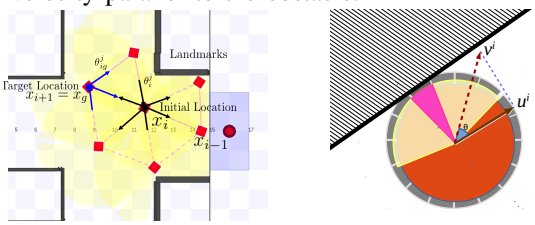
C. Control of Robots

We use the visual homing controller described in Section II-D (Figure 10(a)). For a frontier robot, i, the desired bearings $\theta^i_{j,\text{des}}$ can be computed easily for the planned direction for deployment of the new robot (θ^j_{new} in the current coordinate frame if robot i), and assuming that the robots are separated by a distance of $r_v - \epsilon$. For every other robot, i', on the path through which robots are being "pushed", $\theta^{i'}_{j,\text{des}}$ are the current bearing values for the robot ahead of i' in the path (with correct ID re-orderings performed).

When robots are being "pushed" along a path, multiple robots move simultaneously, and for a robot moving on the path, some of its landmarks (*i.e.*, neighbors) are themselves moving. The bearing-based controller that we use, is in fact capable of dealing with moving landmarks, and give similar convergence properties. A few static landmarks (at least two in total) referenced by some of the moving robots on the path are sufficient in attaining convergence. In addition, the desired bearing is set for each robot for all of their surrounding neighbors. This allows robots to adaptively correct their trajectory while simultaneously gaining and loosing landmarks along their trajectory.

No robots reference the robot that is moving to a new (unexplored) location for expanding the frontier. This is because there are uncertainties about the unexplored region (*e.g.*, about presence of obstacles), and errors due to that should not propagate upstream. Furthermore, if a robot does not have more than one another robot to reference to as landmark, it employs an open-loop control to reach the desired location using odometry estimate, and drives back in case it loses the single visual link that it had. This is unavoidable when, for example, the robots move in a narrow passage in a single file.

1) Action on Touching an Obstacle: Upon touching an obstacle at a bearing of $\theta^i_o \pm \tau$ (τ being the resolution in the measurement of bearing to touch), the robot will not be able to progress in the direction between $(\theta^i_o - \frac{\pi}{2} + \tau, \theta^i_{jo} + \frac{\pi}{2} - \tau)$ (Figure 10(b)). Hence, if the command velocity, v^i, computed using the bearing-only controller "pushes" the robot inside an obstacle, we take the best projection of that velocity into the set of allowed velocities (u^i in the figure, falling inside the brown sector) such that using u^i as the velocity command the robot moves out toward the obstacle-free area freeing itself from the obstacle. Overall, this results in a behavior akin to sliding along the obstacle using the component of the velocity parallel to the obstacle.



(a) The bearing-based controller uses neighbors as landmarks and use the bearing angles to them to navigate to the desired location knowing the desired bearing angles.

(b) Upon touching an obstacle, the robot use the component of the computed velocity that is the projection in the valid/free sector (brown).

Fig. 10. Components of the controller.

2) Scale Correction: Since our controller is purely bearing-based, and although we attempt to create a hexagonal packing, small accumulation of the errors can decrease the average separation between the robots as we move further away from the source. To correct this, we perform a scale correction periodically, where we make a frontier robot, i, move forward keeping the reference robots behind it (opposite to a mean bearing to those robots), until it breaks visual link with at least one of those neighboring robots. Then we make the robot i drive back until it reestablishes the visual link with all its neighbors. This ensures that the average separation between the robots stay in $O(r_v)$.

D. Identification and Reallocation of Redundant Robots

Using the method described in [11] and briefly discussed in Section II-E, we can identify redundant robots in the complex by computing a generator (non-trivial relative cycle) of the relative homology $H_2(\mathcal{R}_{r_v}, \mathcal{K})$, where $\mathcal{K} = \mathcal{F} \cup \mathcal{O}$

is the *fence* subcomplex. We use the JavaPlex [21] library for the computation of the non-trivial relative cycle using persistence algorithm. The required filtration over $\mathcal{R}_{r_v}$ is achieved by inserting the 0, 1 and 2 dimensional simplices in sequence. We add a disjoint 0-simplex, Q, as illustrated in Figure 5(b), and construct the 1-simplices $\{i, Q\}$ and the 2-simplices $\{i, j, Q\}$, for every 0-simplex, i, in $\mathcal{K}$, and every 1-simplex, $\{i, j\}$, in $\mathcal{K}$. Call this new complex $(\mathcal{R}_{r_v}, \mathcal{K})$. Computation of persistent homology up to dimension 2 for this complex with $\mathbb{Z}_2$ coefficients using JavaPlex gives us a set of non-trivial generating 2-cycles in $(\mathcal{R}_{r_v}, \mathcal{K})$ which generate $H_2(\mathcal{R}_{r_v}, \mathcal{K})$. Any non-zero linear combination (in $\mathbb{Z}_2$ coefficients) of these cycles will also be a valid non-trivial 2-cycle which can be used to identify the robots that are sufficient for maintaining coverage. Thus we perform a greedy search for the *best* linear combination (a linear combination of cycles such that it contains the least number of 0-simplices) that also contain all the fence 0-simplices.

Thus, finally we have a set of 0-simplices which constitute robots that are sufficient to maintain the sensor coverage that is currently being maintained. All other robots are redundant and can be removed/reallocated. Once we have identified the redundant robots, in the next deployment cycle we use them, instead of deploying new ones from the source.

IV. Guarantees

Since throughout the deployment and covering process we keep the graph (1-skeleton of $\mathcal{R}_{r_v}$) of the already-covered region fixed (we only *"push"* robots along paths in the graph to the frontiers), we eliminate the possibility that the algorithm gets stuck in an infinite cycle in which the graph keeps cycling/switching between two configurations. Furthermore, by choosing to keep the graph structure fixed across deployment cycles, we eliminate the possibility that our control algorithm results in recession of a frontier or opens up a new hole in the already-covered region of the environment. If due to accumulation of errors we do open up a hole, and hence a new set of frontier 1-simplices appear, we send robots to those frontier 1-simplices to fill the hole.

Algorithm Termination: The algorithm, as described, will keep deploying robots to frontier 1-simplices as long as they exist. In absence of obstacles nearby, a robot will be deployed for every frontier 1-simplex at an angle of 60° with the simplex into the uncovered region. This will always make the frontier progress (as illustrated in Figure 9). Although this may result in deployment of redundant robots (which are later identified and removed using the relative H_2 homology generator computation), the progress in the expansion of the frontier is always finite in obstacle-free regions. Nevertheless, when the expanding location lies inside an obstacle we need to consider, and thus avoid, the possibility that robots are deployed indefinitely to a region close to an obstacle because the unexplored region changing only infinitesimally at each deployment. This is however prevented by the design of our algorithm, as described in Section III-A item 'iv.', where we prevent the deployment of unnecessary robots near obstacles that make little to no progress in expanding the frontier. Thus, in a finite environment the algorithm will terminate with no more frontiers left for exploration.

Algorithmic Completeness: As described, when $\mathcal{F}$ is not empty, more robots will be deployed to close the frontier. When $\mathcal{F}$ is empty, then one can observe that for every 2-simplex $\{i, j, k\}$ in $\mathcal{R}_{r_v}$, the convex hull of the robots i, j and k will be covered by the disk of visibility for each of these robots. If $\{i, j\}$ is a 1-simplex in $\mathcal{O}$, then due to the way we introduce elements in $\mathcal{O}$ (Section III-A), the region between the obstacle 1-simplices and the actual obstacles themselves will always remain covered by some robot's disk of visibility (except for non-convex features on the side of the obstacles that are smaller than r_v). Thus, when $\mathcal{F}$ is empty, we can guarantee sensor coverage of the entire environment.

Robustness to Robot Failure: The proposed algorithm can adapt to failure of robots. If a robot fails (and its neighbors can detect that), the swarm will ignore the presence of the failed robot. Thus a "hole" in the complex gets created, and hence $\mathcal{F}$ will have the frontier simplices surrounding that hole. This will result in new robot(s) being deployed to the newly open frontier, until $\mathcal{F}$ becomes empty once again.

Optimal Coverage: While our deployment algorithm itself does not guarantee optimality, the process of identifying redundant robots by computing the smallest non-trivial relative cycle in $(\mathcal{R}_{r_v}, \mathcal{K})$ (described in section III-D), and hence redistribution of the redundant robots, makes sure that we do not use more roots than required to cover the entire environment. While this still is not a guarantee of global optimality, this indeed is a local optimality in the sense that after redistribution we end up with a complex that is the optimum subcomplex of the original complex without any redundant robot.

Limitations: Since the robots use the omnidirectional cameras only to obtain bearing to neighbors, and the only way they sense obstacles is through direct touch/contact, it is not possible to detect features that are smaller than r_v. Thus, presence of non-convex features on the surface of the obstacles that are smaller than r_v may result in some "blind-spots" inside the non-convex "grooves".

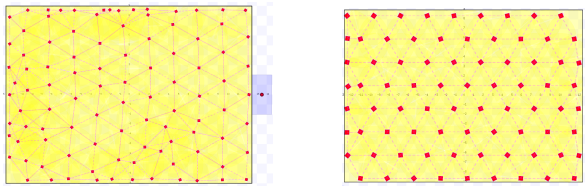
V. Results

We demonstrate the performance of the proposed algorithm in simulation using integrated platform between Robot Operating System (ROS) [22], Stage Simulator [23] and JavaPlex [21]. We use ROS as a backbone that links all components together. Stage simulates the dynamics and sensors of the robots, while JavaPlex is used to compute the relative homology for identifying the redundant robot.

A. Comparison with Hexagonal Packing

To evaluate the performance of the proposed algorithms, we compare the number of robots used in covering an obstacle-free rectangular region using our algorithm and using the hexagonal packing, which we constructed manually by overlaying the environment on a hexagonal packing in a free space. In an obstacle-free environment, the performance of our algorithms is comparable to the hexagonal packing solution as illustrated in Figure 11. The majority of the

robots deployed by our algorithm are in the hexagonal packing arrangement. However, due to accumulated errors and collision with the boundary, the packing gets distorted and the clutter of robots is higher near the boundary.



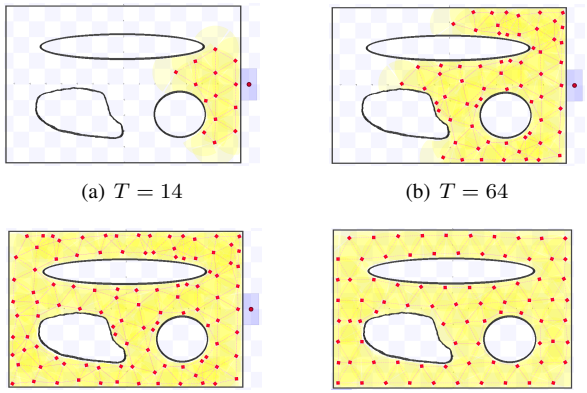
(a) Deployment using the proposed algorithms (robots deployed from the source on the right).

(b) Hexagonal Packing using the same average separation between the robots, r_v.

Fig. 11. Our algorithm deployed 98 robots while the hexagonal packing requires approximately 78 robots.

B. Structured environment

Our algorithm attains a similar performance in a structured environment with few obstacles as illustrated in Figure 12. Comparison between figures (c) and (d) illustrates that the performance of our algorithm is comparable to the hexagonal packing. We present more results in cluttered environments in the video accompanying this paper.



(a) $T = 14$

(b) $T = 64$

(c) $T = 106$

(d) Hexagonal Packing

Fig. 12. Demonstration in a structured environment with obstacles. Figures (a)-(c) illustrate the progress of our sensor coverage algorithm at $14, 64$, and 106 deployment cycles respectively. Figure (d) is the "ideal" hexagonal packing in the environment for comparison, attained using 82 robots and using the same average separation between the robots.

VI. Discussions

In this paper, we proposed an algorithm for the deployment of a swarm of resource-constrained, mobile robots in an unknown environment with the objective of attaining complete sensor coverage of the environment without using any metric information. The only sensors are a limited range omnidirectional camera that can detect bearing to neighboring robots and a touch sensor for detecting contact with obstacles and other robots. No global information is available. The proposed algorithm, which is derived from concepts in algebraic topology, is complete, terminates in finite environments, is robust to noise and robot failures, and is locally optimal. We demonstrate the performance of the proposed algorithm through a C++ implementation on a ROS platform

References

[1] IA Wagner, M. Lindenbaum, and AM. Bruckstein. Distributed covering by ant-robots using evaporating traces. *Robotics and Automation, IEEE Transactions on*, 15(5):918–933, Oct 1999.

[2] Ercan U. Acar, Howie Choset, Alfred A. Rizzi, Prasad N. Atkar, and Douglas Hull. Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21(4):331–344, 2002.

[3] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258 – 1276, 2013.

[4] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Trans. Robot. Autom.*, 20(2):243–255, April 2004.

[5] Subhrajit Bhattacharya, Robert Ghrist, and Vijay Kumar. Multi-robot coverage and exploration on riemannian manifolds with boundary. *International Journal of Robotics Research*, 33(1):113–137, January 2014. DOI: 10.1177/0278364913507324.

[6] Samuel Rutishauser, Nikolaus Correll, and Alcherio Martinoli. Collaborative coverage using a swarm of networked miniature robots. *Robotics and Autonomous Systems*, 57(5):517 – 525, 2009.

[7] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *Robotics and Automation, IEEE Transactions on*, 7(6):859–865, Dec 1991.

[8] Sven Koenig, Boleslaw Szymanski, and Yaxin Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):41–76, 2001.

[9] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[10] Maxim Batalin and Gaurav S. Sukhatme. The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment. *IEEE Transactions on Robotics*, 23(4):661–675, Aug 2007.

[11] V. de Silva and R. Ghrist. Coordinate-free coverage in sensor networks with controlled boundaries via homology. *The International Journal of Robotics Research*, 25(12):1205–1222, 2006.

[12] R. Ghrist, D. Lipsky, J. Derenick, and A. Speranzon. Topological landmark-based navigation and mapping, 2012.

[13] Jason Derenick, Vijay Kumar, and Ali Jadbabaie. Towards simplicial coverage repair for mobile robot teams. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5472–5477. IEEE, 2010.

[14] A. Muhammad and M. Egerstedt. Control using higher order lapla cians in network topologies. In *Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems*, pages 1024–1038, Kyoto, Japan, 2006.

[15] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: A survey. In *Proceedings of the 2008 Conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 363–371, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[16] SeoungKyou Lee, Aaron Becker, Sándor P. Fekete, Alexander Kröller, and James McLurkin. Exploration via structured triangulation by a multi-robot system with bearing-only low-resolution sensors. *CoRR*, abs/1402.0400, 2014.

[17] Allen Hatcher. *Algebraic Topology*. Cambridge Univ. Press, 2001.

[18] R. Tron and K. Daniilidis. Technical report on Optimization-Based Bearing-Only Visual Homing with Applications to a 2-D Unicycle Model. *ArXiv e-prints*, February 2014.

[19] H.-C. Chang and L.-C. Wang. A Simple Proof of Thue's Theorem on Circle Packing. *ArXiv e-prints*, September 2010.

[20] M. Sghaier, H. Zgaya, S. Hammadi, and C. Tahon. A distributed dijkstra's algorithm for the implementation of a real time carpooling service with an optimized aspect on siblings. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 795–800, Sept 2010.

[21] Andrew Tausz, Mikael Vejdemo-Johansson, and Henry Adams. Javaplex: A research software package for persistent (co)homology. Software, 2011.

[22] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, Kobe, Japan, May 2009.

[23] The stage robot simulator. http://rtv.github.io/Stage/.