Article

The International Journal of Robotics Research
2016, Vol. 35(10) 1234–1260

DOI: 10.1177/0278364915624974
ijr.sagepub.com

Structured triangulation in multi-robot systems: Coverage, patrolling, Voronoi partitions, and geodesic centers

Seoung Kyou Lee[1], Sándor P. Fekete[2] and James McLurkin[1]

Abstract

We present a fundamental framework for organizing exploration, coverage, and surveillance by a swarm of robots with limited individual capabilities, based on triangulating an unknown environment with a multi-robot system. Locally, an individual triangle is easy for a single robot to manage and covers a small area; globally, the topology of the triangulation approximately captures the geometry of the entire environment. Combined, a multi-robot system can explore, map, navigate, and patrol. Algorithms can store information in triangles that the robots can read and write as they run their algorithms. This creates a physical data structure *(PDS) that is both robust and versatile.*

We study distributed approaches to triangulating an unknown, two-dimensional Euclidean space using a multi-robot network. The resulting PDS is a compact representation of the workspace, contains distributed knowledge of each triangle, encodes the dual graph of the triangulation, and supports reads and writes of auxiliary data. The ability to store and process this auxiliary information enables the simple robots to solve complex problems. We develop distributed algorithms for dual-graph navigation, patrolling, construction of a topological Voronoi tessellation, and location of the geodesic centers in non-convex regions. We provide theoretical performance guarantees for the quality of constructed triangulation and the connectivity of a dual graph in the triangulation. In addition, we show that the path lengths of the physical navigation are within a constant factor of the shortest-path Euclidean distance. We validate these theoretical results with simulations and experiments with a dozen or more robots.

Keywords

Multi-robot system, distributed algorithm, space coverage, robotic network, localization, navigation

1. Introduction

Consider an unknown environment, such as a factory after a fire, a building after an earthquake or a landscape following a flood. How can we make use of mobile robots in order to explore the area, construct a useful map, and respond to ongoing developments? How can we secure the area, so that new events or developments are quickly detected and a rapid response can be dispatched? And how can we structure exploration and surveillance in the first place, before a disaster even happens?

Fundamental problems arising from these kinds of scenarios have been considered from a variety of different angles, with many impressive scientific and practical achievements. Work on individual mobile robots has produced platforms that are capable of amazing feats; however, even an extremely powerful robot cannot be in several places at once, therefore it cannot detect parallel events, has to travel between locations, and is vulnerable to catastrophic failures. On the other hand, the field of sensor networks has been able to develop a wide range of methods and technologies for achieving distributed coverage, parallel sensing and processing, and robustness; however, these platforms are immobile, requiring deployment beforehand and precluding flexible response to dynamic developments.

This is where systems of many simple robots can offer the best of both worlds: they combine mobility and flexibility with parallelism and robustness. The basic idea is to deploy a mobile sensor network that informs other robots about the surrounding environment, where we envision deploying a heterogeneous group of robots. A large number of small, low-cost, simple robots initially spread out to map the environment and sense events. With proper organization, we can use their topological map as a discrete representation of the surrounding environment. Thus,

[1]Department of Computer Sciences, Rice University, USA
[2]Department of Computer Science, TU Braunschweig, Germany

Corresponding author:
Sándor P. Fekete, Department of Computer Science, TU Braunschweig, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany.
Email: s.fekete@tu-bs.de

we can provide support for many kinds of tasks, allowing us to achieve localization of sensory events, distributed computation, and support for other, more capable, robots, without requiring global information, communications, or processing.

There are several crucial challenges that must be solved in order to reach this objective. How can we organize a swarm of simple robots in order to explore an unknown environment? How can we use active deployment as a stepping stone for building structures that continue to be useful and flexible for surveillance and guarding, such that the advantages of distributed and parallel action are not wiped out by the impediments of limited sensing and bounded local information and coordination? And how can we use the structured swarm for even higher-order tasks?

This paper provides an integrated framework for solving this hierarchy of challenges, combining swarm robotics with ideas from sensor networks as well as analytic methods from algorithmics. At the first level, our approach is based on constructing a "good" triangulation of the environment during exploration, even with very simple and weak individual robots. We demonstrate that triangulating the workspace with a multi-robot system is a useful stepping stone for a wide range of tasks, even when using robots with limited sensors and sensor accuracy. As it turns out, these capabilities suffice to achieve triangulation with provable trigonometric performance guarantees, even in the worst case. To this end, the simple robots are positioned such that they become the vertices of a triangulation of the environment. The triangulation provides complete coverage, can be built using basic local geometry, allows proofs of properties for coverage and navigation, and provides distributed computation and data storage.

Making use of the underlying topological structure of a triangulation is also the key to handling the second challenge: the triangles in the triangulation create a *physical data structure* (PDS) of computational elements embedded in the physical world. Exploiting the dual graph of a triangulation for distributed mapping and navigation leads to provable performance guarantees.

Considering computation on the triangles instead of the robots provides a useful abstraction that eases computation. At the third level, we use the triangles to compute a discretized approximation of a Voronoi partition, called the topological Voronoi tessellation (TVT). Using the TVT, the triangles can then compute the geodesic center of their Voronoi partition. A straightforward approach to this computation is susceptible to errors from local minima. Using the triangles as computational elements allows us to introduce *virtual agents*, simulated robots that perform computation and travel from triangle to triangle at the speed of message communication. These virtual agents find all the minima, and then separate the local from the globally optimal center. A mobile robot positioned at this center can respond to events in their TVT partition with optimal response time.

All three levels of this framework (Sections 4, 5 and 6) are presented in a modular manner. For all algorithmic solutions, we present technical assumptions, theoretical results, algorithms, simulation results, and hardware experiments, demonstrating the integration of provable analytic results with practical simulations and experiments.

Our contributions

Research on particular aspects of exploration, mapping, navigation and surveillance of known and unknown environments is a central topic in; (1) robotics; (2) sensor networks; and (3) algorithmics (in Section 2, we describe a selection of related work from all three fields). In particular, making use of swarms of many simple components instead of single, more powerful robots has received a growing amount of attention. However, the vast majority of previous research has been carried out separately.

1. Impressive work from robotics has been largely heuristic, without provable theoretical performance guarantees.
2. Sensor networks are able to carry out a wide range of distributed computing tasks, but only provide a stationary infrastructure that lacks many of the key features of mobile robots.
3. While algorithmics has managed to achieve many sophisticated methods, these are often only founded in theory.

The key contribution of our work is to develop an overall framework for a (potentially huge) swarm of simple, mobile robots that combines ideas and methods from all three fields, and manages to blend them into one integrated, multi-level approach that achieves results that provide; (1) provable theoretical performance guarantees; (2) a robust sensory infrastructure that combines aspects of stationary sensor networks with mobile robotics; (3) practically useful and experimentally validated methods.

We present in detail a multi-level, fully distributed approach that is structured into three stages.

1. At the deployment level, we develop simple distributed exploration methods for a swarm of simple robots in unknown environments, based on triangulation requiring only local information and geometry. The key idea is that even very limited capabilities in sensing and control suffice to construct a triangulation of "good" trigonometric properties, which achieves provably good coverage. The corresponding module is described in Section 4, "triangulation construction in unknown space".
2. At the navigation level, we show how the achieved triangulation can be employed as a physical data structure for navigation and routing. The key idea is that in a good triangulation, basic dual graph navigation achieves provably good performance in the original geometric space, even with very limited sensing capabilities and without

global coordination. Details of this module are provided in Section 5, "navigation in triangulated space".

3. At the global infrastructure level, we demonstrate how the benign properties of good triangulations can be used for more advanced structuring, without requiring global computation. The key idea is that we can exploit higher-order topological Voronoi tessellations in order to coordinate more complicated distributed tasks, such as patrolling by a swarm of mobile robots. Section 6, "topological Voronoi tessellation", presents details of the resulting module.

For each of these modules, we describe the underlying theoretical and technical assumptions, establish provable theoretical worst-case performance guarantees, provide details of technical implementation, and present experimental validation and evaluation.

2. Related work

There has been a large body of work in the field of robotics on using robot swarms for navigation and exploration. Here we only mention the work most closely related to ours, in particular those employing some elements of sensor networks. Notable is the approach by Batalin and Sukhatme (2004), which is similar to our own in that it combines mobile robots with a sensor network, but uses estimated transition probabilities between nodes to compute the best direction to suggest to a mobile robot for moving between a start and goal node. These transition probabilities are established during deployment, with both the robots and sensor nodes having synchronized direction sensors. McLurkin and Smith (2004) present a breadth-first distribution from a more practical view, using a swarm of 100 robots. However, their work forms a sensor network that does not contain the environment's geometric features. Another bridge between robotics and sensor networks is described by McLurkin and Demaine (2009); they use ideas similar to the ones for detecting the boundary of a stationary sensor network (see Fekete et al. (2004), Fekete et al. (2005), Kröller et al. (2006)) for identifying the boundary of a robot swarm. Burgard et al. (2000) coordinated multiple robots using a probabilistic method to explore unknown space, but assumed that each robot knows the relative positions of all other robots. This differs from our work that pursues a fully decentralized network, so that each robot is able to share local information only with its immediate neighbors that are within communication range.

Konolige et al. (2006) and Ko et al. (2003) designed distributed algorithms for multi-robot localization and mapping in an unknown environment. However, their work allows the loss of communication between a pair of robots while exploring an unknown environment. Partial maps collected by each robot can only be merged after two robots rendezvous at the same location and share their map data. If disconnection happens, one robot in a rendezvous position cannot announce its current location to other disconnected robots. As a result, the robot at the rendezvous location only relies on the "chance" to meet with other robots by waiting at its current location. This method cannot deterministically guarantee the algorithmic correctness without employing more advanced methodologies for achieving a rendezvous; even though there is a wide range of models and methods for dealing with rendezvous search, this is a deep theoretical field (Alpern and Gal, 2003).

Another highly interesting work is the GNAT project by O'Hara et al. (2005), which aims at building a sensor network that interacts with mobile robots to support various applications. This, however, does not consider the dispersion of a large number of sensors in an unknown environment. Instead, the authors focus on the device structure of each sensor and the way in which a sensor network interacts with mobile robots. Unless these are provided *a priori*, this also precludes enabling provably good properties of the employed sensor network, which is one of the key features of our work.

Other extremely relevant work is presented by Rutishauser et al. (2009), who placed a group of miniature robots to cover a target space using a collaborative scheme. While this also provides distributed methods for dispersing robots uniformly, it uses a different avenue for dealing with the geometry of the environment, giving robots the individual ability to localize themselves based on a predefined, global color-coding scheme. This differs from our approach, which does not require or provide any such prior knowledge (or global coordinates such as GPS). Durham et al. (2010) present a pursuit-evasion algorithm for a team of robots, whereas Ghoshal and Shell (2011) and Nouyan and Dorigo (2006) describe an RRT-like algorithm. Those works use the robot's physical positions as a data structure to store intermediate results of the algorithm. On the other hand, robots used by Durham et al. (2010) are not stationary. Therefore, sensing an event in these robot networks becomes time sensitive. Moreover, random-based branch expansion in a tree-like structure by Ghoshal and Shell (2011) and Nouyan and Dorigo (2006) may degenerate the space coverage efficiency because of unbalanced robot density. Other exciting recent work on formations of massive swarms of robots is presented by Rubenstein et al. (2014), who describe how to achieve geometric shapes with simple robots. However, the simplicity of the employed kilobot platform requires some external control, so distributed methods for achieving guaranteed geometric properties appear to be elusive. Instead, algorithmic approaches take a quite different angle (e.g. as described in Becker et al. (2013a), Becker et al. (2014) and Becker et al. (2015)).

In contrast to physical data structures (PDSs) for navigation in a triangulated space, our approach allows us to use triangles as computational elements, which provide useful geometric properties and support practical distributed computations for robot navigation and exploration using only local information and communications. Approaches such as those used by Spears et al. (2004) and Turgut et al. (2008)

build a triangle lattice using potential fields, but this is not a *triangulation*: there is no knowledge of triangles, the dual graph, or distributed data structures for computation, so the robots never recognize that they form a triangulation. Therefore, many studies, including Geraerts (2010) or Kallmann (2005), use a triangulated environment for path planning, but require global information and localization.

For triangulation construction, we combine the ideas of routing in stationary sensor networks (Fekete and Kröller, 2006, 2007; Kröller et al., 2009) with approaches for dynamic robot swarms. New challenges arise when considering a large number of real-life mobile nodes with limited capabilities. The online minimum relay triangulation problem (MRTP) has received attention from the theoretical side by Fekete et al. (2011), who show that a competitive factor of 3 for the number of necessary nodes can be achieved, whereas no constant factor can be guaranteed for the related online maximum area triangulation problem (MATP) of maximizing the space of a given number of nodes. For settings with axis-parallel boundaries of the environment, see Fekete et al. (2013). The MATP is revisited in Section 4.2, in which we demonstrate that certain geometric assumptions can allow such a constant factor for the online MATP. In a more general setting, triangulations with shape constraints for the triangles and the use of Steiner points are considered in mesh generation, see the survey by Bern and Eppstein (1992). With respect to the particular capabilities of the robot, we remark that our work has been motivated by and evaluated on a particular platform (the r-one), but works similarly well for others, as long as some basic trigonometry can be performed (e.g. when being able to measure distances instead of angles). In this context, the interested reader is directed to the survey by Suri et al. (2008) on the abilities of robot exploration with minimal sensing.

The problem of placing a minimum number of relays with a limited communication range in order to achieve a connected network (a generalization of the classical Steiner tree problem) is considered by Efrat et al. (2008), who give a number of approximation results for the offline problem (a 3.11-approximation for the one-tier version and a PTAS for the two-tier version of this problem). A similar question is considered by Bredin et al. (2010), who reported on the minimum number of relays needing to be placed to assure a k-connected network. They present approximation results for the offline problem. Kashyap et al. (2011) concentrate on approximation algorithms for the case of $k = 2$ in higher-dimensional Euclidean space. Asking for multipath connectivity in a network increases the fault tolerance against node failures. We aim for a well-connected network, but the triangulated structure allows other applications. Moreover, k-connectivity approaches do not consider whether an entire region is covered, or how much of the area can be covered. For swarms, Hsiang et al. (2004) consider the problem of dispersing a swarm of simple robots in a cellular environment, minimizing the time until every cell is occupied by a robot. For workspaces with a single entrance door, Hsiang et al. present algorithms with time optimal *makespan* and $\Theta(\log(k+1))$-competitive algorithms for k doors. Maintaining cohesiveness of a robot swarm in the presence of exterior forces is the subject of a recent paper by Krupke et al. (2015).

With respect to topological Voronoi tessellation, we place a set of robots at their optimal positions for the minimum worst-case response time. The proposed algorithms adapt an approach similar to Lloyd's algorithm with Voronoi tessellation; however, we face a more difficult scenario, because we preclude global positioning or maps of an environment. Cortes et al. (2004) used Lloyd's algorithm with a density function around the target region, such as a light source, to deploy robots around the target with resting density. This work, however, uses the standard Voronoi diagram, and only guarantees convergence when the Voronoi regions are convex. Several authors have extended Lloyd's algorithm to operate in non-convex regions. This makes it necessary to deal with the case in which the centroid is not within the navigable region, but instead is inside a concavity, or inside of an obstacle. Bhattacharya et al. (2013) computed the closest point on the boundary of the region to the centroid, then moved the robot there. Breitenmoser et al. (2010b) presented a similar approach, but used the tangent bug algorithm to find the closest point. Both approaches do not guarantee the location with the minimum response time. Moreover, these works require either self-localization in shared coordinate or an external source for global positioning. Yun and Rusy (2012) proposed a distributed vertex substitution algorithm to compute Voronoi centroids for multiple robots in non-convex and discrete space, computing a configuration that is *locally* optimal. Breitenmoser et al. (2010a) used a triangle mesh to compute centroids Voronoi tessellation of a non-planar surface. They assumed that each robot can get the mesh information within the communication range and share it with neighbor robots, therefore the result highly depends on the length of mesh edges.

A more rigorous approach is to compute the *geodesic center* of the Voronoi region. Aronov (1989) presented an algorithm to compute a geodesic Voronoi tessellation in non-convex environments, but did not compute the geodesic center. It is possible to compute the geodesic center in a brute-force fashion by considering each point in the region, but this is computationally intractable for large regions. Many approximations can be found in the robotics literature. Durham et al. (2012) described a pairwise gossip algorithm that makes robots converge to the center of Voronoi cells constructed on a discrete grid; however, they require a map with a unified coordinate system. Moreover, their work allows that each Voronoi region can be larger than the sensing or communication radius of a robot. This requires each robot in each Voronoi cell to physically move to adjust each Voronoi region and to sense an event. Therefore, it may not be possible to sense multiple events immediately and in parallel, even though multiple robots are employed.

Pimenta et al. (2008) used objective function with various types of distance including the geodesic distance for geodesic center in a non-convex environment and the *power distance* for heterogeneous types of robots, but allowed a centralized approach. For instance, two robots in their work are able to interact with each other to generate a Voronoi cell, even when the distance between them is larger than the radius of robots' sensing footprints. For this reason, they used an objective function in a global coordinate system. Note that our paper aims at a fully distributed approach, where no external device, map or global information is allowed. Caicedo-Nuez and Zefran (2008) applied a diffeomorphism that transforms a connected non-convex region to a convex region to run Lloyd's algorithm appropriately, which, however, requires complete knowledge of an environment.

Finally, we mention that some of the ideas in this paper have been presented in condensed and preliminary form in the conference abstracts Lee et al. (2014a) and Lee et al. (2014b); see Becker et al. (2013b) for a video illustrating some of the foundations of online triangulation, i.e. the results presented in Section 4.

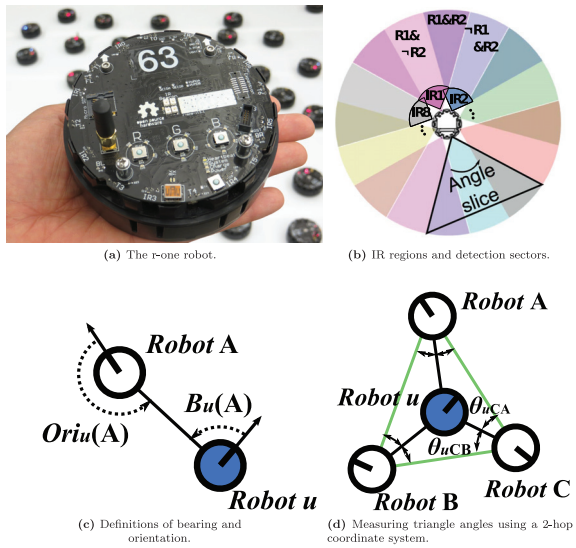
3. Model and preliminaries

We have a system of n triangulation robots and p navigation robots. The communication network is an undirected graph $G = (V, E)$. Each robot is modeled as a vertex, $u \in V$, where V is the set of all robots and E is the set of all robot-to-robot communication links. The neighbors of each vertex u are the set of robots within line-of-sight communication range r_{max} of robot u, denoted $N(u) = \{v \in V \mid \{u, v\} \in E\}$.

During construction, we ensure that all network edges, except the edge facing a wall, are also navigable paths. In other words, the minimum edge length of a triangulation exceeds the diameter of a navigation robot, so that the navigation robot is able to pass through any edges while navigating triangle by triangle. Robot u sits at the origin of its local coordinate system, with the $\hat{x}$-axis aligned with its current heading. Robot u cannot measure distance to its neighbors, but can only measure the *bearing* and *orientation*, as shown in Figure 1c.

We assume that these angular measurements have limited resolution. Note that these assumptions are based on a real-world platform: all experiments in this paper use r-one robots, shown in Figure 1a. This robot is equipped with 8 IR receivers that form $\frac{\pi}{8}$ radian bearing angle resolution, as shown in Figure 1b. Clearly, our overall approach remains valid for other platforms with similarly limited capabilities, provided that the underlying trigonometric properties can be achieved (for example, being able to approximately measure distances instead of angles works in a similar manner).

Robots share their angle measurements with their neighbors. In this way, robot u can learn of all angles in its 2-hop neighborhood. Figure 1d shows the relevant *inner angles* of a triangle around u. Each neighbor of u computes these angles from local bearing measurements, then announces them. The communication used by these messages is $O(\max(\delta(u) \in V)^2)$, where $\delta(u)$ is the degree of vertex u.



(a) The r-one robot. (b) IR regions and detection sectors. (c) Definitions of bearing and orientation. (d) Measuring triangle angles using a 2-hop coordinate system.

Fig. 1. (a) (Foreground) the r-one robot is an advanced, low-cost, open-source platform designed for research, education, and outreach. **(b)** 16 IR communication sectors by 8 IR-transmitter and 8 IR- receiver sensors. Each sector has limited resolution of $\frac{\pi}{8}$. **(c)** Robot u can measure the bearing to neighbor A, $B_u(A)$, and the orientation of neighbor A, $Ori_u(A)$. **(d)** Triangle angles (black arrows) are measured from neighbors of robot u, and shared with u using a local broadcast message.

Each robot has contact sensors that detect collisions with the environment. There is an obstacle avoidance behavior that can effectively maneuver the robot away from these collisions. The robots also have a short-range obstacle sensor that can detect walls closer than $\approx$ 50 cm. The obstacle sensor does not detect neighboring robots.

Algorithm execution occurs in a series of synchronous *rounds*, t_r. This greatly simplifies analysis and is straightforward to implement in a physical system, as mentioned in the work by McLurkin (2008). At the end of each round, every robot u broadcasts a message to all of its neighbors. The robots randomly offset their initial transmission to minimize collisions. During the duration of each round, robot u receives a message from each neighbor $v \in N(u)$. Each message contains a set of public variables, including the sending robot's unique ID number $u.id$ and other variables defined later. Note that the a message at each round has a constant size. Each message contains a fixed number of variables, where most of them use fixed number of bits (uint8 or uint16), except one variable to express the ID of a message sender. For the ID variable, the number of required bits increases as we use more robots. However, the size is also bounded by $log_2 n$, i.e. the number of bits to express n distinctive ID numbers. Moreover, this work regards $log_2 n$

as constant because the MATP problem considers a fixed number of robots to triangulate the maximum area. Therefore, the size of each message becomes constant because all variables in the message use a bounded number of bits.

4. Triangulation construction in unknown space

In this section, we present a distributed approach to construct a triangulation in an unknown environment. If the number of available robots is not bounded *a priori*, the problem of minimizing their number for covering all of the region is known as the *minimum relay triangulation problem*(MRTP); if their number is fixed, the objective is to maximize the covered area, which is known as the *maximum area triangulation problem* (MATP). Each of these problems has been studied, both for the *offline* scenario (in which the region is fully known), and the *online* scenario (where the region is not known in advance) (Fekete et al., 2011; Lee et al., 2014b). The online MRTP admits a 3-competitive strategy, while the online MATP does not allow a bounded competitive factor: if the region consists of many narrow corridors, we may run out of robots exploring them, and thereby miss a large room that could permit large triangles. Later in this paper, we will argue that practical considerations allow us to overcome this bound in realistic scenarios. In a more theoretical setting (in which the region that is to be explored consists of unit pixels, making all corridors at least 1 wide), Fekete et al. (2013) provided competitive factors slightly above 1.5. In this section, our method for triangulating is based on the MATP, however in the rest of this paper, we assume that the population of robots is large enough to triangulate the entire workspace.

4.1. Online max-area triangulation algorithm

From the definition of the online MATP, the purpose of the max-area triangulation algorithm is to triangulate the maximum area of an unknown environment using a fixed number of robots. Our assumptions preclude the use of global localization or a geometric model of the environment. Therefore, it is not possible to use a centralized approach that directly computes the optimal positions of robots for the globally maximum area of triangulation. Instead, we apply an incremental approach for the online MATP algorithm. Each robot enters a currently triangulated workspace sequentially, navigates in the triangulation, creates new triangles in the untriangulated region that extends the current triangulation. This process continues until the entire environment has been triangulated.

Figure 2 illustrates the execution of the online max-area triangulation algorithm. We assume that two *base robots* initially mark the *base edge* - such as a door to an unexplored building, where all other robots are behind the base edge. The algorithm starts with this base edge and proceeds by constructing a triangulation in a breadth-first manner. The triangulation is extended as robots construct triangles along the current *frontier* of exploration. The frontier is shown as blue lines in Figure 2, and it delineates the boundary between triangulated space and untriangulated space. All the area between the base edge and the frontier is triangulated. Each mobile robot extends the frontier by moving into unexplored space and forming a triangle with itself and at least two other adjacent robots from the frontier. The algorithm terminates when either all of the workspace has been explored, or the maximum number of robots has been exhausted. Each robot tries to build a *high-quality* triangle - one that does not have edges that are too short or angles that are too small. Equilateral triangles are ideal, but cannot always be constructed due to errors or environmental constraints. We use two properties of the resulting structure, lower bounds on edge length and angles, to provide theoretical performance guarantees. We formally define this quality metric in Section 4.2. In a practical implementation, the minimum edge length is bounded by the robot size, and the maximum edge length is bounded by the communications range of the robots. The angular properties are limited by the type of sensor used, and the limits of the motion controller to drive to the equilateral points of the new triangle.

During algorithm execution, we distinguish the following types of edges in the robot network G;

1. *Frontier edges* (blue lines in Figure 2), $\{u, v\} \in E_F$, which belong to only one triangle and have at least one vertex that is not in contact with the wall.
2. *Wall edges*, $\{u, v\} \in E_W$, which also belong to only one triangle, but both vertices of the edge are in contact with a wall. Both frontier edges and wall edges lie on the exterior of a triangulated network, but a navigating robot only passes through frontier edges.
3. *Internal edges*, $\{u, v\} \in E_I$ which belong to two adjacent triangles.

We do not allow for any overlapping triangles because each robot creates new triangles by passing through only one frontier edge, and always updates new frontier edges as the edges are faced with an uncovered area. Hence, each edge can only belong to either one or two triangles and all edges that are not a frontier edge, base edge, or wall edge become an internal edge. The dotted yellow lines indicate the dual graph, D, which connects adjacent triangles and can be used for navigation in the triangulated region. We address details of dual graph navigation in Section 5.

Construction of a new triangle begins with the addition of a new *navigating robot*, u. To build the triangulation in a breadth-first fashion, a *frontier triangle* is selected that is the minimum distance in the dual graph from the base triangle. This triangle will have at least one frontier edge; we select it to be the *goal frontier edge*, $\{l, r\}$. The robot uses the dual graph to navigate to the frontier triangle. More details about the navigation algorithm are described in Sections 5.1 and 5.2.

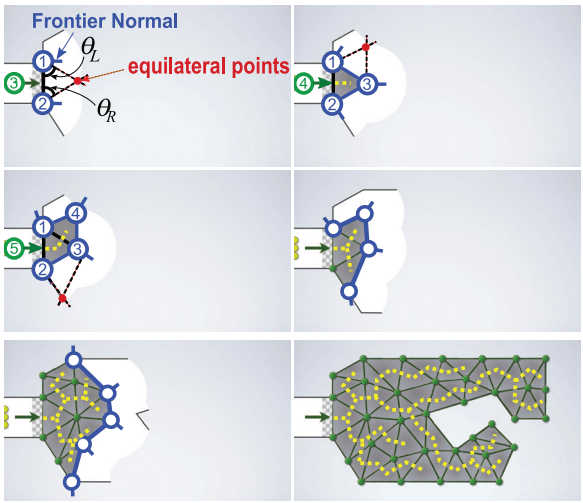


Fig. 2. Constructing a triangulation in a BFS manner. The edges of the BFS tree in the dual graph are dashed yellow, and blue-colored lines denote frontier edges distinguishing covered area from uncovered region. Blue-colored circles are robots that form frontier edges. Among these robots, we refer to a robot in free space as a *frontier* robot and a robot in contact with a wall as a *frontier-wall* robot. The number in each robot denotes the order of a robot entering into a workspace to expand triangles. Each robot colored in green behind the initial edge $\{1,2\}$ subsequently enters into a workspace, creates a new triangle when it arrives at the *equilateral point*, becomes a frontier robot (or a frontier-wall robot), and updates frontier edges. On the other hand, all edges having at least one non-frontier robot are all interior edges colored in green. The blue tick marks on each robot show the direction of the *frontier normal*. This points into unexplored space, in the direction perpendicular to the frontier edges incident at each robot.

A new triangle can be formed in two ways, *expansion* or *discovery*. Figure 3a illustrates the construction of a triangle by expansion. When *navigating robot* u is within the frontier triangle, it switches to the *expanding state*, and moves towards the equilateral point for the new triangle. When u crosses the frontier edge $\{l,r\}$, it creates a new expansion triangle Δulr ($l = l_0$ and $r = r_0$ in Figure 3a). Once robot u arrives at the equilateral point, it switches to the *expanded state*, and adds Δulr to its list of triangles, becoming its owner. Edge $\{l,r\}$ becomes an internal edge, and robot u broadcasts a message to neighbors l and r, so that they update their right and left frontier neighbors to u. This ensures that each robot can only create one expansion triangle, and prevents creating overlapping triangles.

When u enters the expanded state, it needs to discover all of the unexpanded high-quality triangles adjacent to Δulr. Figure 3b shows an example of triangle discovery. We describe the process for the left frontier neighbor (l); it is analogous for the right. We label the left neighbors $\{l_0, l_1, \ldots\}$, where $l_0 \equiv l$. Robot u first considers neighbor l_1, then proceeds through each neighbor on its left side in counter-clockwise order. For each neighbor $l_i, i \geq 1$, robot u checks for edge $\{l_i, l_{i-1}\} \in E_F$. If this edge exists, then u forms a candidate triangle, Δul_il_{i-1} (light green in Figure 3b), and evaluates its quality using definition 4.1. If the candidate triangle is high quality, robot u becomes its owner, and switches its left frontier neighbor from l_{i-1} to l_i. Robot u then broadcasts a message to l_i to update its right frontier neighbor from i_{i-1} to u. Because the edge $\{l,r\}$ is now internal, it is not used for expansion again. Robot u repeats the above procedure to discover other high-quality, adjacent triangles, and can create and own multiple triangles. When complete, u changes its state to *frontier*. Note that all messages used to expand or discover triangles have constant size and are transmitted continually rather than instantaneously, like a token-based protocol. By doing so, we avoid any errors caused by communication failures.

4.2. Covered area

The quality of the triangulation affects the practical and theoretical performance guarantees of our applications, including the patrolling and topological Voronoi algorithms in Sections 5 and 6. Let $r_{\max}$ be the maximum length of a triangulation edge. We also consider a lower bound of $r_{\min}$ on the length of the shortest edge in the triangulation; in particular, we assume that the local construction ensures that any non-boundary edge is long enough to let a robot pass between the two robots marking the vertices of the edge, so $r_{\min} \geq 2\delta$, where δ is the diameter of a robot. The practical validity of these assumptions for a real-world robot platform will be shown in Section 4.4. Finally, angular measurements of neighbor positions let us guarantee a minimum angle of α in all triangles. These constraints give rise to the following:

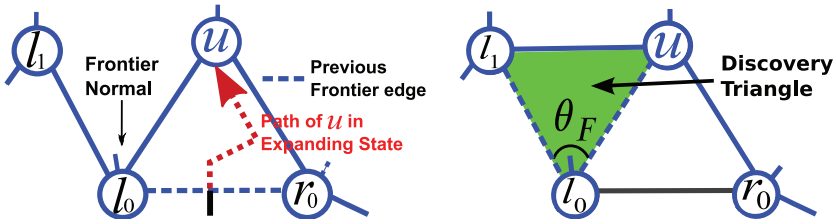
Definition 4.1. *Let $\mathcal{T}$ be a triangulation of a planar region $\mathcal{R}$, with vertex set V. $\mathcal{T}$ is (ρ,α)-*fat*, if it satisfies the following properties;*

1. *The ratio r_{max}/r_{min} of the longest to shortest edge in $\mathcal{T}$ is upper bounded by some positive ρ.*
2. *All angles in $\mathcal{T}$ have size at least α.*

This definition is also used to prove properties of the coverage control based on tessellations in Sections 5.3 and 6.1. Given the definition, we then establish theoretical performance guarantee about the quality of triangulation.

Theorem 4.1. *Consider a (ρ,α)-*fat *triangulation of a set V with n vertices, with maximum edge length r_{max} and minimum edge length r_{min}. Then the total triangulated area is within $\sqrt{3}\rho^2/\sin(\alpha)$ of the optimum.*

Proof. In a (ρ,α)-*fat* triangulation, each edge has length between $r_{\min}$ and $r_{\max}$, and any angle is bounded from below by α. Then suppose that the triangulation has $|T|$ triangles. The largest possible area covered by the triangulation cannot exceed the covered area by a triangulation with $|T|$ equilateral triangles whose edge length is $r_{\max}$. In this optimum



(a) Forming an *expansion* triangle. (b) Forming a *discovery* triangle.

Fig. 3. **(a)** An example *expansion* triangle. The red arrow shows the path of robot u in state *expanding*. After arriving at the equilateral point, robot u switches to the *expanded* state, and the triangle is complete. **(b)** An example *discovery* triangle is shown in light green. Robot u checks θ_F to evaluate the quality of candidate triangle, $\Delta u l_1 l_0$. If this triangle is sufficiently high quality, the robot u stores $\Delta u l_1 l_0$ in its triangle list.

case, the area of the triangulation is $|T|\sqrt{3}r_{\max}^2/2$. On the other hand, the minimum possible area covered by the triangulation happens when the triangulation is composed of $|T|$ triangles, where each of them covers the minimum area. From elementary trigonometry, an obtuse triangle has area $(\sin\alpha)\, r_{\min} r'/2$, where $r' \geq r_{min}$. Hence, the covered area by an obtuse triangle is at least $(\sin\alpha)\, r_{\min}^2/2$. Likewise, an acute triangle has area at least $(\cos\alpha/2)\, r_{\min}^2/2$. Because $\alpha \leq \pi/3$, we have $(\cos\alpha/2) \geq \sin\alpha$ and the minimum area covered by the triangulation becomes $|T|(\sin\alpha)\, r_{\min}^2/2$. Therefore, the maximum ratio of the covered areas with a given (ρ,α)-*fat* triangulation becomes $\frac{|T|\sqrt{3}r_{\max}^2/2}{(|T|\sin\alpha)r_{\min}^2/2} = \sqrt{3}\rho^2/\sin(\alpha)$. □

Note that in a practical setting, ρ will be much smaller than the worst theoretically possible case. See Figure 12 for a real-world evaluation.

4.3. Implementation

A high-level finite-state machine for the MAT algorithm is shown in Figure 4. Two robots are initialized in the state *frontier-wall* and placed at the base-edge. All other robots begin behind the base edge in state *navigation*. Table 1 lists the helper functions in the algorithms below.

4.3.1. Navigation-internal state. The navigation contains three states; *nav-internal*, *expand-triangle*, and *wall-follow*. A new robot, u, enters the network in the *nav-internal* state, and runs algorithm 1 to navigate to a frontier triangle. Line 2 runs an occupancy test function, shown in Figure 5a, that returns the current triangle, T_c, which contains the robot u, and its owner, o. If T_c is not a frontier triangle, then the robot u moves to an adjacent triangle that is topologically closer to a frontier triangle using the navigation algorithm described in Section 5.2.

If T_c is a frontier triangle (line 3) or null (only true if the robot u has just crossed the base edge, line 6), then the robot u will create a new triangle. The variables $u.L$ and $u.R$ are set to the left and right neighbors of the frontier edge (lines 4 and 7), and the robot changes its state to *expand-triangle* (lines 5 and 8).

Algorithm 1 NAV-INTERNAL

```
1:  while u.state =Navigate-Internal do
2:      T_c ← GETCURRENTTRIANGLE()
3:      if ISFRONTIERTRIANGLE(T_c) then
4:          (u.L, u.R) ← GETFRONTIEREDGENBR(T_c)
5:          u.state ← Expand-Triangle
6:      else if ISONLYBASEEDGE(N(u)) then
7:          (u.L, u.R) ← GETBASEEDGENBR()
8:          u.state ← Expand-Triangle
9:      else
10:         T_next ← GETCLOSERADJTRI(T_c)
11:         MOVETONEXTTRIANGLE(T_next)
12:     end if
13: end while
```

Algorithm 2 EXPAND-TRIANGLE

```
1:  while u.state = Expand-Triangle do
2:      (θ_L, θ_R) ← GETINNERANGLE(u.L, u.R)
3:      TRIANGLEEXPANSIONCONTROLLER(θ_L, θ_R)
4:      if ISINGOALREGION(θ_L, θ_R) then
5:          STORETRIANGLESTOLIST (Δuu.Lu.R)
6:          T_D ← DISCOVERTRIANGLE(u.L, u.R)
7:          UPDATEFNBR()
8:          BCASTFMSG()
9:          BCASTDISCONNECTMSG()
10:         STORETRIANGLESTOLIST (T_D)
11:         u.state ← Frontier
12:     else if ISWALLDETECTED() then
13:         u.state ← Wall-Follow
14:     end if
15: end while
```

4.3.2. Expand-triangle state. In the state *expand-triangle*, a robot u runs algorithm 2. Line 2 computes the left and right inner angles to the frontier neighbors, θ_L and θ_R. Line 3 runs the triangle-expansion controller illustrated in Figure 5b until the robot u is in region 3.

Instead of a complete description of the controller, we briefly sketch its operation here. When the robot u enters region 3, if $\theta_L > \theta_R$, the robot u first moves toward $B_u(u.L)+\pi$ until $\theta_R \geq \frac{\pi}{3}$, where B_u denotes the bearing

Table 1. Table of helper functions.

GETCURRENTTRIANGLE()	Runs occupancy test and returns current triangle, T_c.
GETCLOSERADJTRI(T_c)	Gets T_c's adjacent triangle that is topologically closest to any frontier triangles.
GetInnerAngle($u.L, u.R$)	Computes and returns left and right inner angle, θ_L and θ_R.
DISCOVERTRIANGLE($u.L, u.R$)	Runs discovery procedure and gets discovery triangles, T_D, and list of u's old and new frontier neighbors.
ISISOSCELESTRIANGLE($u.L, u.R$)	Checks if $\theta_L = \theta_R$ in an expand triangle.
GETFRONTIERWALLNBR($u.L, u.R$)	Returns $u.L$ or $u.R$ in frontier-wall state.
BCASTFMSG()	Broadcasts new frontier msg to nbrs.
RECVFMSG()	Receives new frontier nbrs.
UPDATEFNBR()	Changes frontier nbrs of u.
BCASTDISCONNECTMSG()	Broadcasts disconnect msg to nbrs.
RECVDISCONNECTMSG()	Returns u_{sender} if u_{sender} disconnects u.
ISCONTAINFRONTIEREDGE(T_i)	Checks if T_i has a frontier edge.
UPDATETRIANGLEHOP(T_i,$N(o)$))	For each triangle u owns, sets its hop to 1 + minimum among all adjacent triangles' hops.
BCASTTRIANGLEHOP($N(o)$)	Broadcasts hops of all triangles u owns.

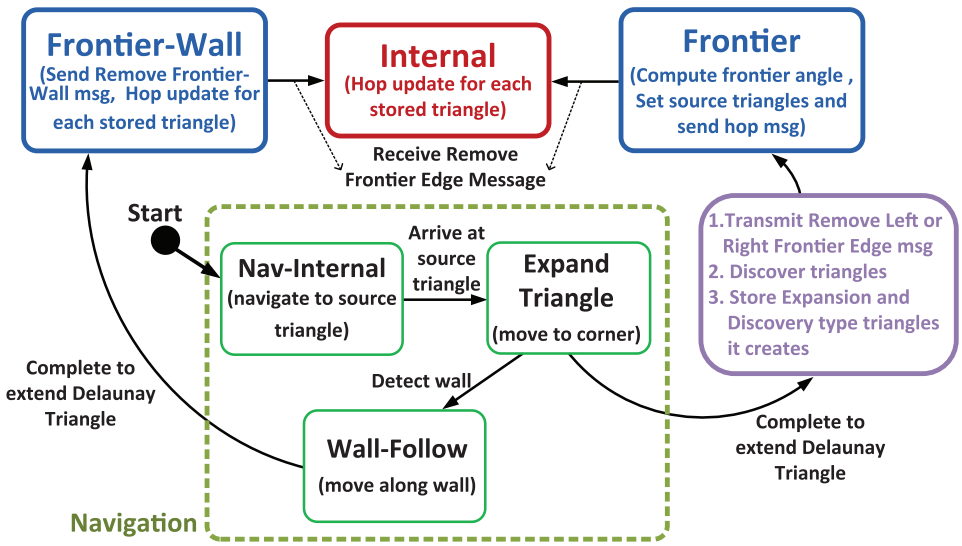
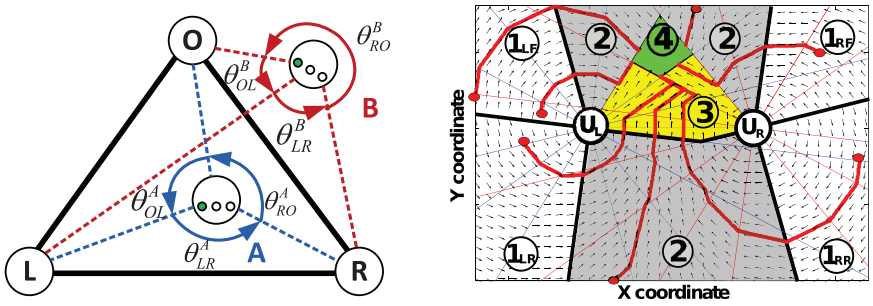


Fig. 4. Finite-state machine for the MAT algorithm. A robot behind a base line enters a workspace and initially moves to state *nav-internal*. Note that if there are no triangles in the workspace yet, a robot directly transits to state *expand triangle*. In the *nav-internal* state, the robot moves through the triangulated space until it reaches a source triangle that has at least one frontier edge. If the robot arrives at the source triangle, it changes its state to the *expand triangle* state, and expands a new triangle into the uncovered region. If the robot successfully expands a new triangle without detecting a wall, the expansion is complete, and the triangle is likely to be high quality (see Figure 7). However, if the robot detects a wall while creating a new triangle, it cannot create a high-quality triangle. So, it changes its mode to state *wall-follow* and follows along the wall until it forms a triangle of the best possible quality. In either case, after expanding a new triangle, the robot discovers adjacent triangles by checking their quality, updates frontier edges, stores the information of its expanded and discovered triangles, and then changes its state to *frontier*.

angle of the neighbor from the robot u. It then changes its heading toward $B_u(u.R)+\pi$, and moves until it reaches the goal region (region 4). The opposite control happens when $\theta_L < \theta_R$.

The robot u stores the triangle on its list (line 5) and runs the *discover triangle* procedure to discover all adjacent triangles (line 6). The *frontier angle*, θ_F, provides a simple way to evaluate the quality of candidate triangles; we define a triangle to be of high quality if $\theta_F < k$, with k manually tuned to reduce errors. Lines 6-11 in algorithm 2 are also related with updating the frontier edges; details are described in Section 4.3.6. On the other hand, if the robot u detects a wall while expanding a triangle (line 12), it changes its state to *wall-follow* (line 13).

4.3.3. Wall-follow state. Because a wall prevents the creation of a triangle with the best quality, a robot u then



(a) Triangle occupancy. (b) Triangle expansion controller.

Fig. 5. **(a)** The occupancy test algorithm determines if a robot is inside a given triangle. If any angle between neighbors of u is greater than π, then u is outside of the triangle. **(b)** Diagram of triangle expansion controller regions between robots u_L and u_R, each with $\frac{\pi}{8}$ bearing resolution. A robot in region 1 rotates around u_L (if $\measuredangle uu_Lu_R > \measuredangle uu_Ru_L$) or u_R (if $\measuredangle uu_Lu_R < \measuredangle uu_Ru_L$), so that it always ends in region 2. A robot in region 2 always moves in the direction where its two inner angles are decreasing. By doing so, the controller always guides a robot from a lower to an adjacent higher-number region using only θ_L and θ_R. All sample trajectories (red lines) converge to the goal region.

aims to expand the triangle of second-best quality by moving along the wall. There are various ways for defining the involved quality measure; we use an isosceles triangle that has the largest minimum inner angle.

The robot u in state *wall-follow* executes algorithm 3. Once the robot detects a nearby wall whose distance from the robot is too close to expand the best quality triangle, it moves toward the wall (line 3). By doing so, we can create an isosceles triangle whose minimum inner angle is the largest among the inner angles of other isosceles triangles whose vertex does not have contact with the wall. If the robot u arrives at the wall, it moves along the wall until it forms an isosceles triangle (line 5).

Once the robot u expands an isosceles triangle, the robot then stores the triangle, broadcasts messages to $u.L$ or $u.R$ for updating frontier edges, and changes its state to *frontier-wall* (lines 7-12). In particular, the robot u sends a disconnect message to its frontier-wall neighbor. By doing so, an edge between the robot u and the frontier-wall neighbor becomes inactive, and no other robots will try to pass through the edge.

Note that any owner u is connected to the owners of all its adjacent triangles in the dual graph G^D. We show this in theorem 5.2 in Section 5.1. In addition, these *expand-triangle* and *wall-follow* algorithms use messages to disconnect and connect triangles from the frontier path. A configuration with triangles that are not (ρ,α)-*fat* could require many messages, but this is unlikely. Our implementation allows for 6 messages, but we only used a maximum of 2.

4.3.4. Frontier and frontier-wall state. When a robot u enters the *frontier* or *frontier-wall* state, it becomes stationary and runs algorithm 4. In lines 3-7, u labels all of its triangles that include a frontier edge as frontier triangles. In lines 8-9, these triangles become sources for messages to conduct dual graph navigation in Section 5. In lines 11-13, the frontier robots compute and broadcast the *frontier angle*, θ_F, between adjacent frontier neighbors, in the direction of the frontier normal, shown in Figures 2 and 3b. The algorithm transmits two constant-sized messages, so the message complexity is $O(1)$. To compute θ_F, each frontier robot in state *frontier-wall* conducts line 11 in algorithm 4. We use the formula, $\theta_F^{Temp} = \frac{B_u(u.L)-B_u(u.R)}{2}$, where $\theta_F, B_u(u.L), B_u(u.R) \in (-\pi,\pi]$. We compute the final θ_F by normalizing the θ_F^{Temp} to be in $(-\pi,\pi]$. Note that base robots are initially in state *frontier-wall*.

Algorithm 3 WALL-FOLLOW

```
1:  while u.state = Wall-Follow do
2:      if WallBumped() = FALSE then
3:          MoveTowardWall()
4:      else if IsoscelesTriangle(u.L, u.R) = FALSE
        then
5:          MoveAlongTheWall()
6:      else
7:          StoreTrianglesToList (Δuu.Lu.R)
8:          UpdateFNbr()
9:          BCastFMsg()
10:         u_W ← GetFrontierWallNbr(u.L, u.R)
11:         BCastDisconnectMsg(u_W)
12:         u.state = Frontier-Wall
13:     end if
14: end while
```

4.3.5. Internal state. Robot u is likely to eventually become an *internal* robot when a new navigation robot discovers triangles and broadcasts disconnect messages to nearby robots, including the robot u, to update frontier edges. Although robot u becomes the internal robot, it maintains the list of triangles owned by robot u. The key role of the robots in state *internal* is to relay messages for dual graph navigation. They continuously call functions for updating and broadcasting the messages, like the functions

Algorithm 4 FRONTIER/FRONTIER-WALL()

```
1:  while u.state=Frontier OR u.state=Frontier-Wall do
2:      for all T_i ∈ TriangleList do
3:          if IsContainFrontierEdge(T_i) then
4:              SetFrontierTriangle(T_i)
5:          else
6:              ClearFrontierTriangle(T_i)
7:          end if
8:          UpdateTriangleHop(T_i, N(o))
9:          BCastTriangleHop(N(o))
10:     end for
11:     Compute/RotateToNormalVec(B(u.L), B(u.R))
12:     θ_F ← ComputeFrontierAngle
        (B(u.L), B(u.R))
13:     BroadcastFrontierAngle(θ_F)
14:     if RecvFMsg() then
15:         UpdateFNbr()
16:     end if
17:     v ← RecvDisconnectMsg()
18:     if v.state = Frontier then
19:         u.state ← Internal
20:     else if v.state = Frontier-Wall ∧u.state= Frontier-
        Wall then
21:         u.state ← Internal
22:     end if
23: end while
```

shown in line 8-9 in algorithm 4. We omit the detailed pseudocode for the state *internal*.

4.3.6. Maintaining the frontier path graph. To ensure that the frontier subnetwork P_F remains a path, the navigating robot u updates its frontier edges and the edges of its neighbors when it adds new triangles. The frontier is initially only the base edge. When a robot u expands a triangle with $u.L \in E_F$ and $u.R \in E_F$, it updates its internal frontier edges in line 7 of algorithm 2. It then sends a message to $u.L$ or $u.R$ in line 8, telling them to disconnect from each other (removing the previous frontier edge $\{u.L, u.R\}$) and connect their frontier to the robot u. This message is received in line 15 of algorithm 4.

The process for connecting and disconnecting discovered triangles is similar. The robot u updates and broadcasts its frontier neighbors in lines 7-8 of algorithm 2. A *frontier* or *frontier-wall* neighbor receives the message in line 15 of algorithm 4, disconnects $\{L_i, L_{i-1}\}$ from each other, and connects L_i to the robot u. In this case, the robot u also broadcasts a disconnect message to the robot L_{i-1} (line 9 in algorithm 2 or line 11 in algorithm 3). The L_{i-1} that gets the disconnection message switches to state *internal* (lines 17, 19, and 21 in algorithm 4). This continues for all of $L_i \in N(u)$, eventually leaving the robot u connected to the last L_i, thus preserving the path. The operation on the right neighbors of $N(u)$ is analogous.

A special case is if the robot u enters the state *frontier-wall* and its left (or right) is also in *frontier-wall*; then the frontier endpoint is disconnected, u disconnects from v and v disconnects from the robot u, making the robot u the new endpoint of P_F. The robot v then changes its state to *internal* (line 21 in algorithm 4). The disconnected edge $\{u, v\}$ becomes inactive. By doing so, we prevent other navigation robots from navigating through the edge blocked by a wall.

4.4. Triangulation construction experiments

We have performed several hardware experiments, using the r-one robots shown in the work by McLurkin et al. (2013). The capabilities of this platform support the assumptions in our problem statement; each robot can measure the bearings to its nearby robots (with a limited resolution of only $\frac{\pi}{8}$), and exchange messages including those bearings and necessary information to run an implemented algorithm (in Section 4.3) using inter-robot communication. Each robot also has eight bump sensors that provide wall detection. The interested reader can find a demonstration of some of the basic aspects of the robots in our video (Becker et al., 2013b), along with some real-world footage of building a triangulation. To evaluate a resulting triangulation quality or trace the trajectory of a navigation robot, we use AprilTag (Olson, 2011). This measures the ground-truth position, $P_u = \{x_u, y_u, \theta_u\}$, of each robot u. The robots cannot measure or use the ground-truth position while executing our algorithms. Robots only know the two-hop local network geometry shown in Figure 1d.

Figures 6a and 6b show snapshots of one triangulation experimental trial and the MATLAB analysis. Over 8 trials using 9-16 robots, the average triangulated area was 1.5 ± 0.29 m^2. In addition, it takes 7.8 ± 2.1 robots to cover a unit area (1 m^2). The average of resulting triangulations are (ρ=3.6, α=0.36 rad)-*fat*.

Figure 7a shows that our triangulations cover about 91% of the region behind the frontier edges. The uncovered region is because the top-left and bottom-left corners in Figure 6a are wall edges (incident on two wall robots), and are not expanded by navigating robots. In order to overcome this limitation, navigating robots would need to attempt to expand each wall edge and look for corners. This is inefficient in environments with large perimeters, and impractical with our current hardware, as corner detection is unreliable. Figure 7b shows the distribution of area covered by individual triangles. The initial length of the base edge determines the area of an ideal equilateral triangle to be 0.088 m^2, and our triangles have a mean area of 0.13 m^2, with a standard deviation of 0.065 m^2. This discrepancy is caused by the angle-based sensors; the robots cannot measure range, and therefore cannot control the area of the triangle they produce. We show this by studying individual triangle quality.

Figures 8a and 8b show our measurements of individual triangle quality: the distribution of minimum angle and

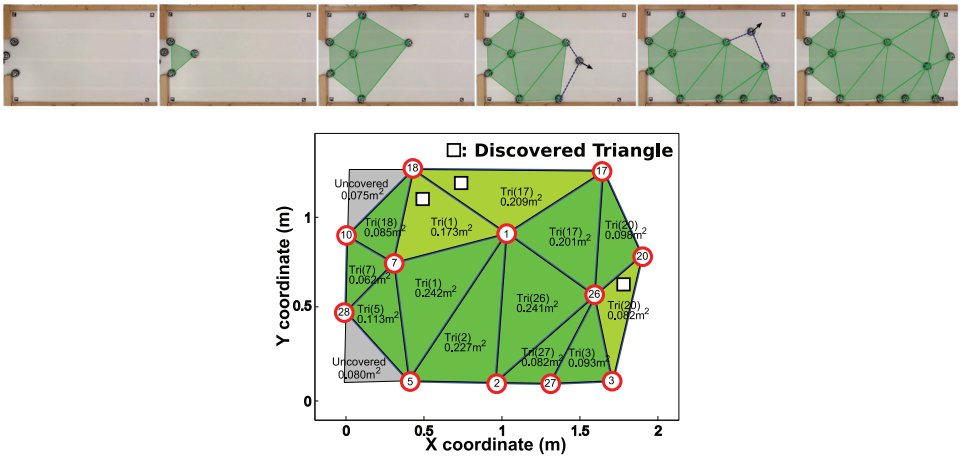


Fig. 6. **(a)** Screenshots while constructing a triangulation with 12 robots. **(b)** MATLAB reconstruction of the resulting triangulation from the physical data structure in Figure 6a. Dark green and light green triangles denote the expansion and discovered triangles. Numbers between parentheses and in the second line are the owner and the size of each triangle. Gray triangles indicate the uncovered region. This is because two robots, 10 and 28, are base robots in state *internal*, and robots 18 and 5 are in state *frontier-wall*, while expanding their triangles. Those edges, {10, 18} and {28, 5}, then become internal edges that do not allow other robots passing through them to expand new triangles.

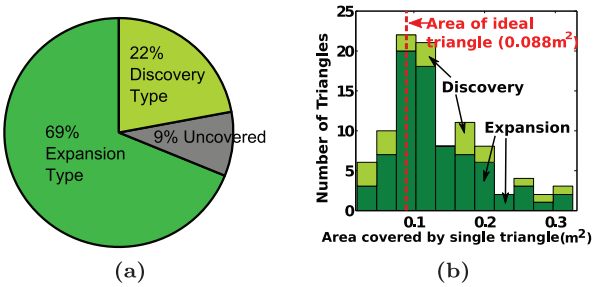


Fig. 7. **(a)** Pie chart for area covered by all triangles. **(b)** Histogram of area covered by each triangle. Dashed red line indicates the area of an equilateral triangle, regarded as ideal shape in our triangulation.

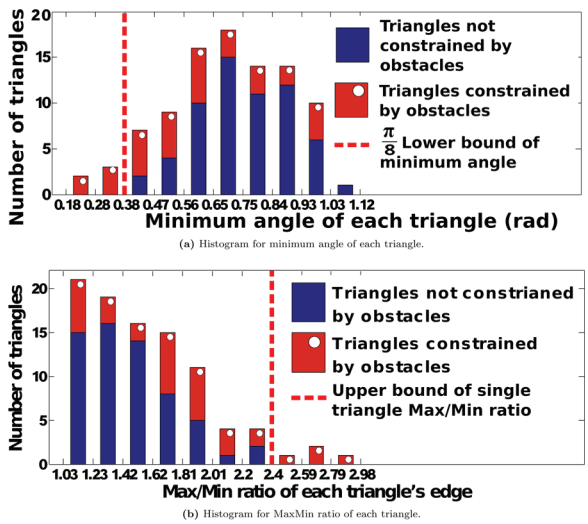


Fig. 8. Histograms of triangle quality in terms of minimum angle (a) and a MaxMin ratio (the ratio between the maximum edge length and the minimum edge length in each triangle) (b) Blue bars are for all triangles created without any disturbance by a wall and red bars are for all triangles interrupted by a wall while expanding. **(a)** Distribution of minimum angle of each triangle, not for a global ρ. All triangles not constrained by a wall satisfy the lower bound, $\frac{\pi}{8}$ (dashed red line). **(b)** Distribution of MaxMin ratio of each triangle. All triangles whose minimum angle is larger than the $\frac{\pi}{8}$ (blue colored bars) also satisfy corresponding upperbound MaxMin ratio.

maximum/minimum edge length ratio (MaxMin ratio) for each triangle. The individual data show triangle quality in a way that overall ρ cannot. An ideal equilateral triangle has a minimum angle of $\frac{\pi}{3}$ rad and MaxMin ratio of one. About 95% of triangles satisfy the lower bound for minimum angle and 96.7% the upper bound for MaxMin ratio. We note that all triangles not constrained by a wall satisfy these bounds, meaning they are approximately the correct shape, but not always the correct size. Knowing range would let us address this, but it is unclear how robots expanding the triangulation should choose between making a triangle of the correct shape or the correct size. In addition, we may be able to implement a more parallel execution method for triangulation construction, where multiple robots navigate through a triangulated region and expand multiple triangles. We leave these for future work.

5. Navigation in triangulated space

After the triangulation is constructed, each robot owns its set of expanded or discovered triangles. This distributed data storage forms a physical data structure (PDS) that

approximately maps the surrounding environment, and creates a dual graph of the corresponding triangulation. In this section, we use the PDS to build a navigation algorithm that can guide a robot to any given target triangle from anywhere in the network.

5.1. Dual graph construction

The dual graph of our triangulation, $G^D = (V^D, E^D)$, where V^D is the set of all triangles stored in all the robots, and E^D is the set of all edges between adjacent triangles.

However, there is no centralized authority that can explicitly keep track of a dual graph, as there are only "primal" vertices, i.e. robots. Our solution is to establish and maintain the dual graph implicitly, by assigning each triangle Δ to a unique robot "owner", $o(\Delta)$, and then mapping edges between triangles in the dual graph, E^D, to edges between robots in the primal graph, E. This mapping will let us reason about relaying messages using the dual graph, and abstract away from the primal graph network. We can use this to realize global objectives, such as routing.

We observe that all owners are connected because all robots in our network, with the exception of the base robots, are owners by construction; every new navigating robot becomes the owner of at least one constructed triangle. A robot can own multiple discovered triangles, and then must maintain multiple vertices in the dual graph.

We must ensure that owners of two triangles connected by an edge in the dual graph can communicate with each other through the primal graph. By doing so, each triangle is able to share its information with all adjacent triangles using only zero or one hop communication to conduct various applications, including dual-graph navigation in Section 5, distributively. If two adjacent triangles Δ_1 and Δ_2, connected through a dual graph edge $\{o(\Delta_1), o(\Delta_2)\}_D \in G^D$, are owned by the same robot $o(\Delta_1) = o(\Delta_2)$, these two triangles can interact with each other using zero-hop communication because the owner robot has the information of both triangles in its local memory space. In this case, we can presume that two triangles are connected via the owner robot. On the other hand, if two adjacent triangles Δ_1 and Δ_2 are owned by two different triangle owners $o(\Delta_1) \neq o(\Delta_2)$, we must show that those two owners are directly connected with each other through an edge in the primal graph $\{o(\Delta_1), o(\Delta_2)\}$.

Lemma 5.1. *Consider a triangle Δ_{abc} with edge $\{a, b\} \in E_F$ as shown in Figure 9a. Let $o(\Delta_{abc})$ be the owner of Δ_{abc}. Then $o(\Delta_{abc}) = a$ or $o(\Delta_{abc}) = b$.*

Proof. By contradiction, assume $o(\Delta_{abc}) \neq a$ and $o(\Delta_{abc}) \neq b$. Then consider the state *expanding* for $\Delta_{o(\Delta_{abc})ab}$. Because $o(\Delta_{abc})$ is the owner in the expanded state, $o(\Delta_{abc})$ must have been the navigation robot in state *expanding*. Therefore $\{a, b\}$ was the frontier edge in state *expanding* and $\{a, b\}$ is now the internal edge in the expanded state, a contradiction. □

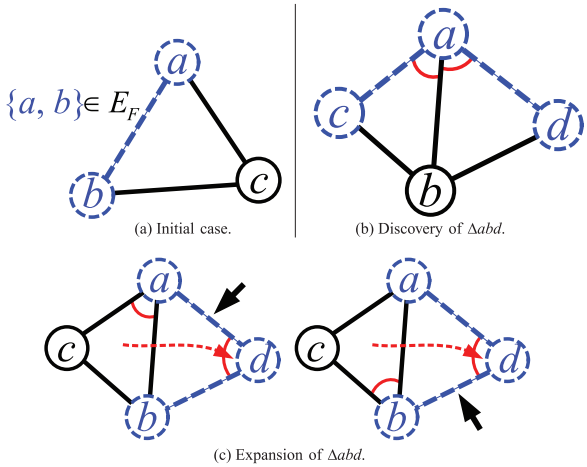


Fig. 9. Illustrations for lemma 5.1. Dashed circles and lines are frontier robots and frontier edges. **(a)** Diagram of lemma 5.1 showing a triangle Δ_{abc}, where the edge $\{a, b\}$ is a frontier edge. **(b)-(c)** Illustrations for theorem 5.2 presenting that two owners of two adjacent triangles are connected. **(b)** If the robot a was a navigation robot in state *expanding*, both triangles are either the expansion triangle or discovery triangle by the robot a. In this case, the robot a is the owner of the triangle Δ_{abc} and Δ_{abd}. **(c)** If the robot d was a navigation robot which expanded the triangle abd, the edge $\{a, b\}$ should be a frontier edge and the robot a or b should be the owner of the triangle Δ_{abc}, from lemma 5.1. Black arrows indicate the edge connecting two owners of the triangles Δ_{abc} and Δ_{abd}. Note that similar logic holds when the robot c was a navigation robot to expand the triangle Δ_{abc} based on the frontier edge $\{a, b\}$).

Given the lemma, we then prove that an owner is located within the communication range of its neighboring owner if their triangles are faced with each other. The proof ensures that an owner of a triangle or a robot inside the triangle is able to acquire the information of all adjacent triangles using only two-hop or fewer communications. In the sense that the information between two adjacent triangles can flow using only distributed communication, we conclude that the dual graph edge is established between two adjacent triangles if, and only if, their owners are connected. This connected dual graph in a triangulation is used for various applications, such as navigable path planning or approximated Voronoi tessellation, with only distributed algorithms.

Theorem 5.2. *The owners of two adjacent triangles must also be connected.*

Proof. Let Δ_{abc} and Δ_{abd} be the two adjacent triangles, and $\{a, b\}$ be the edge they share. As stated in Section 4.1, we do not allow an overlapped triangle, and every edge belongs to at most two triangles. Therefore, these two triangles can be formed in the following two ways (in state *expanding*); □

1. Robot a was the navigation robot. Then a is the owner for both Δ_{abc} and Δ_{abd} as shown in Figure 9b. a is connected to itself.

2. Robot d was the navigation robot. This makes Δ_{abc} an existing triangle and $\{a, b\}$ a frontier edge in state *expanding*. d is also the owner robot for Δ_{abd} in the expanded state. Either a or b is the owner of Δ_{abc} by lemma 5.1, so d, the owner of Δ_{abd}, must be connected to the owner of Δ_{abc} through either edge $\{a, d\}$ or edge $\{b, d\}$ as shown in Figure 9c. By symmetry, b is equivalent to a, and c is equivalent to d.

5.2. Dual graph navigation

We use a dual graph as a navigation guide for robots in a triangulation, resulting in a path shown by the dotted yellow line in Figure 10a. The vertex of a destination triangle becomes the source in the dual graph, sets its hop to zero, and broadcasts messages to build a BFS tree suitable for navigation (Li and Rus, 2005). Each non-source triangle continuously relays broadcast messages to update its hops by setting one more value from the minimum hop among its all adjacent triangles. Resulting hops of all triangles then become the topologically shortest from the destination triangle in a dual graph, called *hop distance*. By theorem 5.2, we guarantee that all triangles are able to compute their valid hop distances from any destination triangle because the dual graph of a triangulation is connected; an owner of a triangle is always connected with all owners of its adjacent triangles.

To move to the destination triangle, a navigation robot first figures out the current triangle that contains the navigation robot using the occupancy test algorithm described in Figure 5a in Section 4.3. The navigation robot then continuously checks the hops of all adjacent triangles of the current triangle, and moves toward the adjacent triangle with the lowest hop distance value. These two steps repeat until the navigation robot arrives at the destination triangle. Note that the navigation robot uses only 2-hop communication to conduct the dual graph navigation algorithm. From the triangle inequality, the owner robot of the current triangle (including the navigation robot) is located in the communication range of the navigation robot. This is because the owner robot can communicate with the rest of the robots forming the triangle, and the communication range of the navigation robot is larger than the owner of the triangle. In addition, theorem 5.2 also ensures that the owner of the current triangle is connected to all owners of its adjacent triangles. Therefore, the navigation robot can acquire the hop distance information of all adjacent triangles by querying the owner robot of the current triangle, who already has the hops of the adjacent triangles using direct communication with the adjacent owners.

As mentioned in the previous section, this method is also used for triangulation construction recall lines 8-9 in algorithm 4 in Section 4.3). Robots owning the frontier triangles set them as source triangles. Each owner, including the owner of frontier triangles, then continuously propagates messages to update the hop distance of all owned triangles. This lets any new robot crossing the base edge move to the topologically closest frontier triangle in the dual graph of a current triangulation, providing a breadth-first construction. This procedure requires only one message from each robot, with $O(1)$ complexity.

5.3. Path stretch

Our previous work shows there is no lower bound on the competitive factor of the stretch of a path in the online MATP problem (Fekete et al., 2011), but this requires narrow corridors of infinitesimal width. As stated in Section 4.2, we made a more realistic assumption that does allow constant-factor performance; we assume the minimum non-boundary edge length is long enough so that a robot can move through any non-boundary edge. Because we also assume network connections are navigable, distances in the dual graph are a discrete approximation of the actual geodesic distance between triangles. Now we can establish that the dual graph of our triangulation can be used for provably good routing. With high-quality triangles, the path length of dual-graph navigation is within a constant factor of the actual distance. We make use of the following terminology.

Definition 5.1. *Consider a connected triangulation $\mathcal{T}$ of a planar region $\mathcal{R}$, with vertex set V, as in Figure 10a. Let s, g be points in $\mathcal{R}$ and let $p(s, g)$ be a polygonal path in $\mathcal{R}$ that connects s to g; let $d_p(s, g)$ be its length. Let Δ_s and Δ_g be the triangles containing s and g, respectively, and let $D(s, g) := \Delta_s, \Delta_1, \ldots, \Delta_\ell, \Delta_g$ be a shortest path in the dual graph of $\mathcal{T}$. Then a* $\mathcal{T}$-greedy *path between s and g is a path $s, q_1, \ldots, q_\ell, g$, such that $q_i \in \Delta_i$, and consecutive vertices of the path are connected by a straight line.*

In other words, a $\mathcal{T}$-greedy path between s and g builds a short connection in the dual graph of the triangulation, and then goes from triangle to triangle along straight segments. Note that we do not make any assumptions whatsoever concerning where we visit each of the triangles.

Lemma 5.3. *Consider a* (ρ, α)-fat *triangle Δ with minimum edge length at least r_{min}; let Δ be intersected by a straight line L. Then the total length of the intersection of L and Δ is at least* $2\sin(\alpha/2)\, r_{min}$ *or the length of the intersection of L with the $r_{min}/2$-disk around one of Δ's vertices is at least* $2\sin(\alpha/2)\, r_{min}$

Proof. See Figure 10b. Consider the closest distance between L and one of the vertices of Δ. If this is larger than $2r_{min}\cos(\alpha/2)$, then we see from Pythagoras' theorem that the intersection of L and Δ must have length at least $2r_{min}\sin(\alpha/2)$. Otherwise the distance is at most $2r_{min}\cos(\alpha/2)$, and the intersection of L with the $r_{min}/2$-disk around the closest vertex of Δ must have length at least $2r_{min}\sin(\alpha/2)$. □

With this, we can proceed to the proof of the theorem for a bound on the stretch factor of resulting paths.

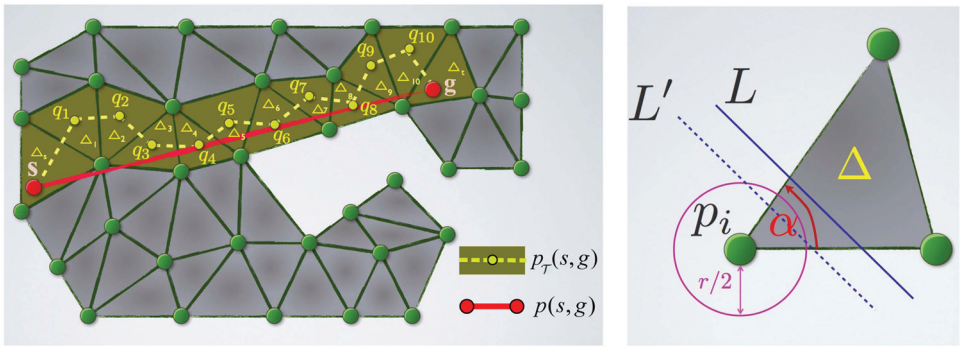


Fig. 10. **(a)** A shortest s,g-path (shown in red) in a region covered by a triangulation $\mathcal{T}$. The resulting $\mathcal{T}$-greedy path is depicted in yellow; a shortest dual path is indicated by colored triangles. Note that each point q_i may be *anywhere* in the respective triangle Δ_i. **(b)** A triangle Δ is intersected by a straight line L. If L passes through the triangle away from one of the endpoints, then the length of the intersection is long. If the line passes through the triangle close to one of the endpoints (indicated by the dashed line L'), then the intersection with a circle of radius $r_{min}/2$ must be long.

Theorem 5.4. *Consider a* (ρ,α)-fat *triangulation* $\mathcal{T}$ *of a planar region* $\mathcal{R}$*, with vertex set* V*, maximum and minimum edge lengths* r_{max} *and* r_{min}*, respectively. Let* s,g *be points in* $\mathcal{R}$ *that are separated by at least one triangle, i.e. the triangles* Δ_s*,* Δ_g *in* $\mathcal{T}$ *that contain* s *and* g *do not share a vertex. Let* $p(s,g)$ *be a shortest polygonal path in* $\mathcal{R}$ *that connects* s *with* g*, and let* $d_p(s,g)$ *be its length. Let* $p_{\mathcal{T}}(s,g)$ *be a* $\mathcal{T}$*-greedy path between* s *and* g*, of length* $d_{p_{\mathcal{T}}}(s,g)$*. Then* $d_{p_{\mathcal{T}}}(s,g) \leq c \cdot d_p(s,g)+2$*, for* $c = \lfloor \frac{2\pi}{\alpha} \rfloor \frac{\rho}{\sin(\alpha/2)}$*, and* $d_{p_{\mathcal{T}}}(s,g) \leq c' \cdot d_p(s,g)$*, for* $c' = \lfloor \frac{6\pi}{\alpha} \rfloor \frac{\rho}{\sin(\alpha/2)}$.

Proof. Consider $p(s,g)$, triangles Δ_s, Δ_g and the sequence $\Delta_1,\ldots,\Delta_{\ell'}$ of ℓ' other triangles intersected by it; by assumption, $\ell' \geq \ell \geq 1$, where ℓ is the number of triangles contained in $p_{\mathcal{T}}(s,g)$. Furthermore, note that the disjointness of Δ_s, Δ_g implies $d_p(s,g) \geq r_{min}$.

We first show that $d_p(s,g) \geq \ell' \lfloor \frac{2\pi}{\alpha} \rfloor 2\sin(\alpha/2) r_{\min}$. For this purpose, we charge the intersection of $p(s,g)$ with Δ_i to Δ_i, if its length is at least $2\sin(\alpha/2) r_{\min}$; if it is shorter, we charge the length of the intersection of $p(s,g)$ with the $r_{min}/2$-disk around one of Δ's vertices p_j evenly to all of the triangles Δ_i that are incident to p_j. Because the minimum angle in a triangle is bounded from below by α, the preceding lemma implies the claimed lower bound on the length of $d_p(s,g)$.

On the other hand, it is straightforward to see that no edge in a T-greedy s,g-path can be longer than $2r_{max}$. Therefore, $d_{p_{\mathcal{T}}}(s,g) \leq 2(\ell+2) r_{max}$. Comparing the lower bound on $d_p(s,g)$ and the upper bound on $d_{p_{\mathcal{T}}}(s,g)$ yields the claim $d_{p_{\mathcal{T}}}(s,g) \leq c \cdot d_p(s,g)+2$ with c as stated. The additive term of 2 results from s and g possibly being close to the boundaries of Δ_s and Δ_g, respectively; it can be removed by noting that $\ell' \geq \ell \geq 1$ implies $(\ell+2) \leq 3\ell'$, as indicated by the second comparison and the choice of c'. □

This provides constant stretch factors even under minimal, highly pessimistic assumptions. The practical performance in real-world settings in which the greedy paths do not visit worst-case points in the visited triangles is considerably better, as we show in Section 5.4.

5.4. Dual graph navigation experiments

We conducted experiments for the dual graph navigation in a triangulation, as shown in Figure 11a. The quality of the triangulation is (ρ=1.36, α = 0.88 rad)-*fat*. Given this triangulated network, we carried out 34 trials for the dual graph navigation from various starting to goal triangle combinations. For each trial, robots forming the triangulation successfully build a breath first tree in the dual graph.

Figure 11b shows the MATLAB reconstruction of the triangulation for the dual graph experiment by combining the data from the PDS of the triangulation with the positioning data of each robot measured by AprilTag (Olson, 2011). Note that the AprilTag positioning data is only for plotting the shape of the triangulation and navigation path. All robots for the triangulation and dual graph navigation do not use this positioning data in these experiments. Other information including triangle ownership, primal graph, and hop of each triangle is from the PDS. The thick blue lines indicate the primal graph that concatenates all pairs of owners whose triangles are adjacent with each other and maps the connected dual graph in the triangulation. Although this graph is not a complete union of triangulation edges, it is a spanning graph of all triangle owners in G, which is implied by theorem 5.2. Five red-colored lines indicate the sample traces of the navigation robot among the entire 34 trials from various initial locations to the same destination triangle. While the shape of most resulting paths is smooth and straight, some parts of them are undulated. These incorrect navigation paths are caused by the errors of the occupancy test algorithm, not by structural errors of the topology in the dual graph. Note that errors of the occupancy test algorithm occur when the navigation robot, whose bearing sensing resolution is $\frac{\pi}{8}$, incorrectly measures the bearing of

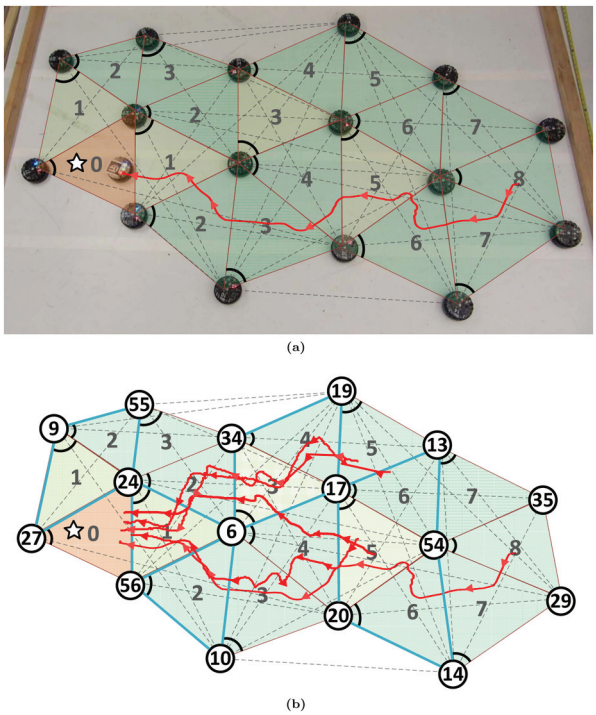


Fig. 11. **(a)** A sample triangulation with 17 r-one robots. A navigating robot used a tree rooted at the red triangle (including a star-shaped marker) to guide the navigation robot from its current location to the goal location, and followed the red-colored trace. The numbers indicate hops in the dual graph from the goal triangle. An arc near each robot denotes triangle ownership. **(b)** Traces of resulting navigation paths using r-one robots. Thick light blue lines indicate primal graph mapping the dual graph of the triangulation. An edge of the primal graph between two owner robots exists if two triangles owned by those robots are adjacent.

its neighbor. Throughout all trials, our occupancy test algorithm returns the correct triangle with a success rate of 91%. In addition, the navigation robot moves to the correct adjacent triangle having the locally minimum hop distance with 99% of the entire triangle transitions.

Figure 12 shows the histogram of the stretch factors for all navigation paths over 34 trials. We measure the length of each navigation path using the odometry embedded in the robot platform used in these experiments. The average and maximum stretch factor are 1.38 ± 0.19 and 1.87. Note that even the maximum stretch factor satisfies the theoretical upper bound stated in theorem 5.4: $1.87 < c' = \frac{6\pi}{0.28\pi} \frac{\rho}{\sin(\frac{0.28\pi}{2})} = 67.8$).

There are practical issues that affect the performance of dual graph navigation. For instance, system faults or message loss may disturb forming a correct hop distribution from specific source triangle. Fortunately, the method used for the dual graph navigation is based on self-stabilized algorithms, such as consensus. Therefore, as long as the system fault or message loss happens temporarily, each triangle in a triangulation converges its hop information from

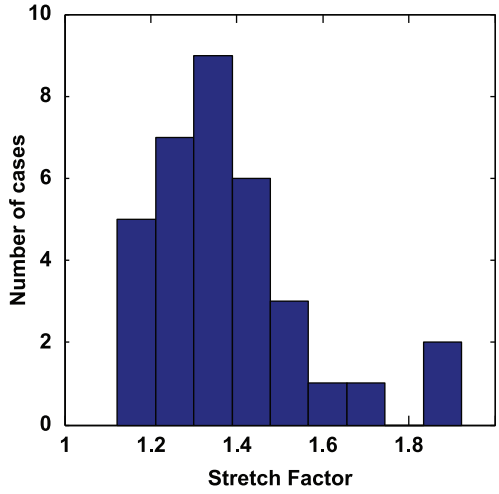


Fig. 12. Stretch factor histogram with 34 trials of navigation experiment.

a target triangle to the correct value after the delay or error is resolved. Note that if the delay or error is not temporary, the hop information never converges to the proper value. A previous study by Fischer et al. (1985) addresses how a faulty process prevents achieving distributed consensus.

In addition, we need to consider the relationship between the performance of dual graph navigation and the quality of a triangulation, the speed of a navigation robot, message broadcast frequency, or the minimum edge length of a triangulation. While navigating in a triangulation, a robot should get hop messages of three adjacent triangles before it transits to another triangle. If the navigation robot is too fast, message broadcast is too slow, or the navigation path is too short, the navigation robot might have to make a decision about "where to go" before receiving hops of all adjacent triangles. In this case, the robot transits to an incorrect adjacent triangle whose hop count is not locally minimal. Suppose that a navigation robot conducts dual graph navigation in a (ρ, α)-*fat* triangulation. Refer to V_R and t_{round} as the velocity of a navigation robot and message broadcast period. From lemma 5.3, the minimum path length to transit from a triangle to another triangle is $2r_{min}sin(\frac{\alpha}{2})$, where r_{min} is the minimum edge length. Hence, the expected time to travel along to the path with velocity V_R will be $\frac{2r_{min}sin(\frac{\alpha}{2})}{V_R}$. Because each robot broadcasts a message within t_{round}, the inequality below should hold in order to guarantee sufficient time for proper dual graph navigation

$$\frac{2r_{min}sin(\frac{\alpha}{2})}{V_R t_{round}} \geq 1$$

For instance, with $V_R = 0.3$ m/s, $t_{round} = 0.25$ round/sec, $r_{min} = 0.6$ m, and $\alpha = 0.88$ rad, used in the above dual graph navigation experiment, the navigation robot takes at least $\frac{2\times 0.6\times sin(0.88/2)}{0.3\times 0.25} = 6.8$ rounds. Therefore, the navigation robot has more than six chances to acquire hop messages while transiting to another triangle. This means that any hop message broadcasting delay (caused by robot system

fault or message collision) within six rounds is allowable for proper dual graph navigation.

5.5. Distributed patrolling

In this section, we apply triangulation to a patrolling application. Patrolling visits all regions in a workspace regularly, with the goal of minimizing the maximum visitation interval, i.e. we wish to visit each region as frequently as possible. Previous patrolling work by Pasqualetti et al. (2012a,b) or Elmaliach et al. (2009) requires global information and complicated computation; our approach is simple and fully distributed. We assume that the patrolling robots move with their maximum velocity. We also assume that the sensing range of each patrolling robot is large enough to cover the largest triangle in the environment. With our triangulated environment, the goal of patrolling can be restated: we desire that the patrolling robots regularly visit all the dual graph vertices in our triangulation. In the following discussion, we refer to triangles as computational elements, but recall that the actual computation is performed by the owner robot.

To measure the performance of our patrolling algorithm, we use the *refresh time*, RT, from Pasqualetti et al. (2012b). For patrolling a triangulation, we modify the definition of the refresh time to the largest *individual refresh time*, $RT_{\Delta_i(t)}$, of each dual graph vertex Δ_i, where the $RT_{\Delta_i(t)}$ indicates the time elapsed since the most recent visitation of any patrolling robot to the triangle i at time t. To update the individual refresh time of each triangle, patrolling robots run the occupancy test algorithm and announce their occupancy to the triangle. The triangle resets its individual refresh time to zero. Once the patrolling robot leaves, the triangle updates the most recent visitation time, t_{last} to indicate when the patrolling robot left. Each triangle can then calculate $RT_{\Delta_i(t)} = t - t_{last}$. Note that each triangle measures the individual refresh time using only the local time frame of its owner. Therefore, we do not require a centralized clock to synchronize time frames between individual triangles.

Given a patrolling robot team in a graph, computing the deterministic team trajectory for the optimal refresh time is NP-hard (it constitutes a traveling salesman problem for one patrolling robot), as stated in the work by Pasqualetti et al. (2012b). Instead, we develop a simple patrolling policy called LRV, for *least recently visited*. This LRV policy continuously guides each patrolling robot to its adjacent triangle having the locally largest $RT_{\Delta_i}(t)$. Figure 13 illustrates an example of adjacent triangles around a patrolling robot. In this local network, the patrolling robot queries the individual refresh times of all adjacent triangles, and moves to the triangle whose individual refresh time is 35. Note that no two (or more) patrolling robots can be in the same triangle. This is because a triangle containing a patrolling robot sets its individual refresh time to the lowest value, 0. This policy uses only local information with no more than two-hop communications and $O(1)$ computational complexity

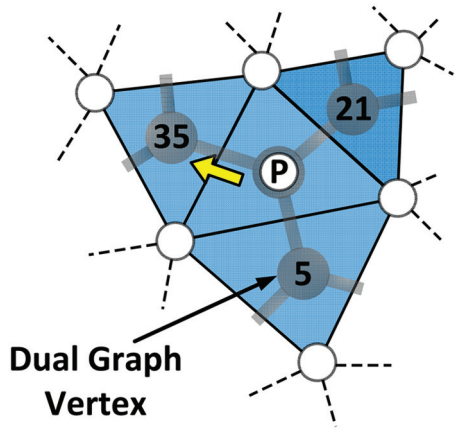


Fig. 13. An illustration of the LRV policy. The numbers indicate the individual refresh time in each triangle. The patrolling robot P collects all individual refresh times of its adjacent triangles, and moves to the adjacent triangle whose individual refresh time is maximum (yellow arrow).

(the maximum number of adjacent triangles is three). While simple, this policy produces complete coverage, as shown by the following theorem; recall that we refer to the dual graph induced by a triangulation as G^D.

Theorem 5.5. *Consider a connected graph $G^D = (V, E)$ with n vertices, patrolled by p robots according to LRV. Then all vertices get visited infinitely often.*

Proof. Let $V_\infty \subseteq V$ be the set of all vertices that get visited infinitely often. Because V is finite, V_∞ is nonempty. Suppose that $V_\infty \neq V$, then the robots must stay within the set V_∞ after some finite time t_0. By definition, they keep visiting all vertices in V_∞, so after $t_1 > t_0$, each of them will have been visited *after time* t_0, for an individual refresh time of $RT_{\Delta_i} < t - t_0$, where $\Delta_i \in V_\infty$. Conversely, all vertices in $V \setminus V_\infty$ will have an individual refresh time $RT_{\Delta_j} \geq t - t_0$, where $\Delta_j \in V \setminus V_\infty$, because they do not get visited again. Because G^D is connected, there must be a vertex $w \in V \setminus V_\infty$ that is adjacent to a vertex $v \in V_\infty$. By definition, v will be visited again; when this happens, the LRV policy ensures that the visiting robot must prefer vertex w (with individual refresh time at least $t - t_0$) to all vertices in V_∞ (with individual refresh times less than $t - t_0$), a contradiction to the assumption that w is not visited again. We conclude that $V = V_\infty$. □

Figure 14b shows the resulting refresh time data using the LRV from the patrolling experiment shown in Figure 14a. The experiment starts with one navigating robot; we add another at 1500 and 3600 seconds. As we deploy more navigating robots, the refresh time decreases, seen in the red line that shows the 400 s moving average. We compute the theoretical lower bound of the refresh time by assuming that all eleven dual graph vertices from the Hamiltonian cycle are evenly divided for the trajectory of each patrolling robot. Note that the resulting refresh time plot never reaches the

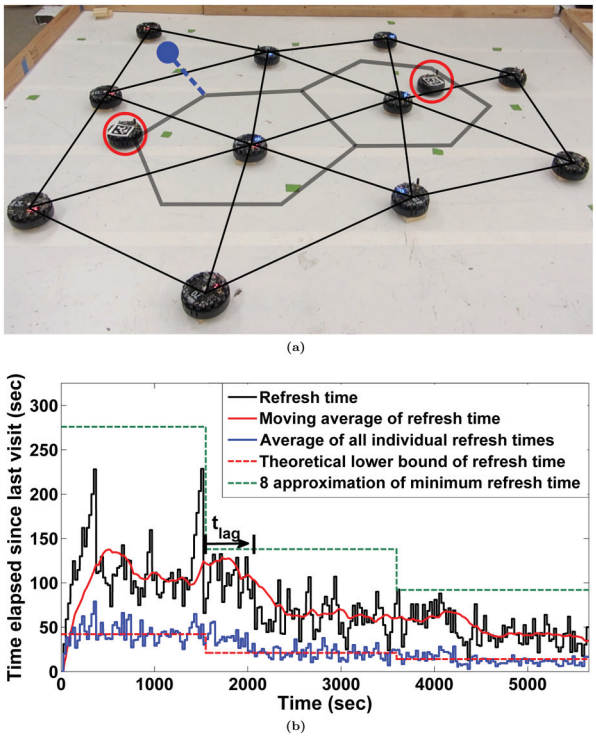


Fig. 14. (a) An experiment setup for LRV experiments with 11 stationary robots. Black lines and red circles depict resulting triangles and patrolling robots, respectively. Dashed blue (at upper left corner) and solid gray lines indicate the dual graph of the triangulation. **(b)** Patrolling experiments result in the triangulation shown in Figure 14a. The black and dashed red lines indicate the refresh time and the theoretical lower bound of the refresh time at each time, respectively. The solid red line is the moving average of the refresh time (window size = 400 seconds) to see the tendency of the refresh time graph. It starts from one patrolling robot, and adds one more patrolling robot after 1500 seconds and 3600 seconds. The refresh time tends to decrease as we add more patrolling robots, but it takes about t_{lag} time to stabilize the refresh time with an increased number of patrolling robots.

theoretical lower bound because the dashed blue edge in Figure 14a is used twice to visit the protruded vertex (blue circle). Therefore, the graph cannot form a Hamiltonian cycle, but the real-world performance is still quite good.

6. Topological Voronoi tessellation

In this section, we use a triangulation to tessellate the workspace into topological Voronoi cells. A conventional method to compute Voronoi tessellation requires a complete notion of an environment that requires a highly centralized approach. A method using a uniformly distributed sensor network by McLurkin and Smith (2004) uses only a distributed method, but may cause a geometrically ambiguous boundary between Voronoi cells, especially if the density of a network is sparse. In this section, we design a fully distributed method that partitions a set of triangles in a triangulated space according to the hop distance in a dual graph, to triangles containing a navigation robot acting as a "site". By doing so, we generate a topological Voronoi tessellation even in a non-convex space. Moreover, Voronoi cells in the resulting topological Voronoi tessellation can form a clear boundary, so that we can combine the topological Voronoi tessellation with various applications, such as patrolling, to enhance the performance of the application.

We assume the environment has been triangulated using an online MATP approach, where a set of triangles discretely approximates the space. Because our triangles are computational elements in the PDS, we model algorithmic execution on the triangles, and rely on the map between the dual and primal graph to assign the computations to the appropriate owner robots. Our robot sensing does not provide distance information about the graph embedding, only angles. However, we can use path length in the dual graph to approximate distances, and then divide the entire triangulated network into *n topologically* equal subnetworks. If we assume that the shape and size of the triangles is similar, we can establish performance guarantees for our results.

An example *topological tessellation* is shown in Figure 15a. We tessellate the environment around p reference points, R_1 and R_2 in the example. Each point is a patrolling robot, constituting the reference point of its topological Voronoi cell. The extent of each cell is the set of closest triangles using geodesic distance. In the implementation, a triangle containing a patrolling robot broadcasts a *cell message*. This message disperses within the graph in a breadth-first fashion, building a tree as it propagates. Triangles without robots join the cell whose source is closest in the dual graph. Triangles equidistant from multiple sources join each cell. This approach uses a message of size $O(1)$ and takes $O(D(G^D))$ rounds of computation, where $D(G^D)$ is the diameter of the dual graph, G^D. Figure 15b presents an experimental result of topological Voronoi tessellation using eleven stationary r-one robots for creating a triangulation and three patrolling r-one robots moving in the triangulation.

6.1. Global path stretch within a region

Theorem 5.4 provides constant stretch factors in the presence of a single robot. In this section, we describe how to provide constant stretch factors even in the presence of multiple robots. Corollary 6.1 addresses performance guarantees for the path stretch *across different cells*, i.e. we show that the topological Voronoi tessellation yields travel distances for the navigation robots (using local information) that are within constant factors of the shortest distance under full information, even if we end up using a navigation robot that is not closest to a target. The proof is largely based on theorem 5.4; again we do not have to worry about navigation between locations that are in triangles sharing a vertex: in that case, local navigation provides good

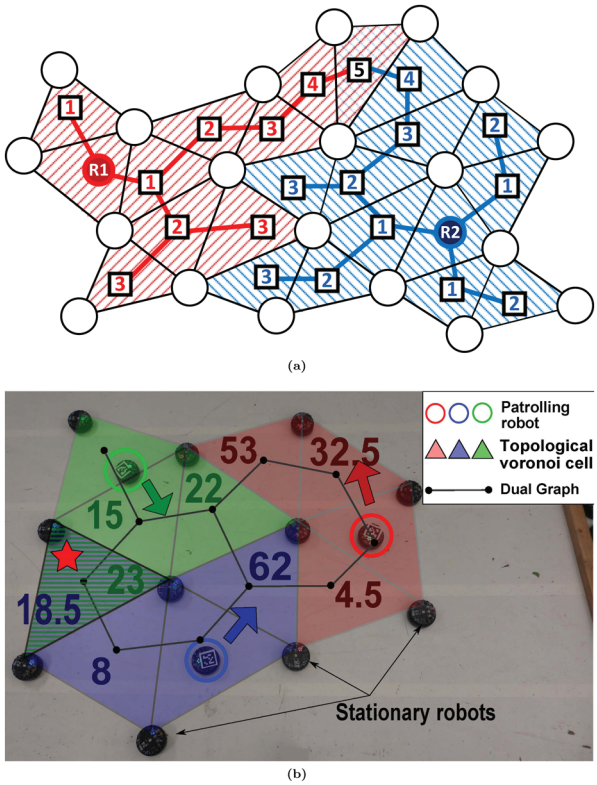


Fig. 15. **(a)** An example of topological Voronoi tessellation. The red and blue circles ($R1$ and $R2$) are the reference points for the tessellation. The triangles that contain them are sources of BFS broadcast trees. The number in each square indicates the hop count in the dual graph to the closest source triangle. Triangles join the cell that is closest, those equidistant join both cells, such as the triangle with five hops. **(b)** Experimental result for a geodesic Voronoi tessellation of a triangulated network with a simple patrolling test. Colored circles and triangles represent source robots and corresponding Voronoi cells. Black dots and lines indicate the dual graph of the triangulation. Note that a triangle that has equal geodesic hops from several source triangles can belong to multiple topological Voronoi cells (leftmost triangle with stripe pattern). The number in each triangle indicates the refresh time for a simple patrolling algorithm.

results. This provides constant stretch factors under minimal, conservative assumptions. The practical performance in real-world settings (where the greedy paths do not visit worst-case points in the visited triangles) is considerably better.

Corollary 6.1. *Consider a (ρ,α)-fat triangulation $\mathcal{T}$ of a planar region $\mathcal{R}$, with vertex set V, maximum and minimum edge length r_{max} and r_{min}, respectively. Let s_1 be the location of a navigation robot, and g be a target location in $\mathcal{R}$ that belongs to the topological Voronoi cell of s_1. Let s_1 and g be separated by at least one triangle, i.e. the triangles Δ_{s_1}, Δ_g in $\mathcal{T}$ that contain s_1 and g do not share a vertex. Then a $\mathcal{T}$-greedy path based on a dual Voronoi tessellation is within a constant factor of a shortest geometric path from any navigation robot to g.*

Proof. By assumption, Δ_{s_1} is the triangle with minimum hop count in the dual graph. On the other hand, suppose s_2 is the location of a navigation robot that is closest to g in geometry. If $s_2 = s_1$, theorem 5.4 provides a proof. Instead, suppose s_2 has a larger hop count in the dual graph, meaning it does not contain g within its cell. Then we can proceed completely analogously to theorem 5.4, but use the lower bound on $d_p(s_2,g)$ and the upper bound on $d_{p_\mathcal{T}}(s_1,g)$ in order to get $d_{p_\mathcal{T}}(s_1,g) \leq c \cdot d_p(s_2,g)+2$, for $c = \lfloor \frac{2\pi}{\alpha} \rfloor \frac{\rho}{\sin(\alpha/2)}$, and $d_{p_\mathcal{T}}(s_1,g) \leq c' \cdot d_p(s_2,g)$, for $c' = \lfloor \frac{6\pi}{\alpha} \rfloor \frac{\rho}{\sin(\alpha/2)}$. □

6.2. Advanced patrolling algorithm using topological Voronoi tessellation

In this section, we develop an advanced policy that shows greater performance than the LRV using topological Voronoi cells. Each robot navigates to the triangle in its cell, V_u, with max $RT_{\Delta_i(t)}$. Figure 16 illustrates the advanced patrolling policy. To compute the refresh time in a cell, the triangles run a leader election algorithm, using individual refresh times as the quantity (McLurkin, 2008). This uses a message of size $O(1)$ and takes $O(D(V_u))$ rounds of computation. Once the maximum individual refresh time triangle is elected, the triangle becomes the source triangle and build a BFS tree in the dual graph of the cell. The patrolling robot then conducts dual graph navigation until it reaches the source triangle.

We compare the performance of the advanced policy with the LRV by analyzing the worst-case scenario. The worst-case refresh time will be proportional to the required hop distance to travel all vertices of a dual graph. Assume without loss of generality that a Voronoi cell, V_u, has the largest diameter $D(V_u)$ and the greatest number of dual graph vertices $|V_u|$ among all cells. Obviously, the inequalities, $D(V_u) \leq n$ and $|V_u| \leq n$, hold, where n is the number of dual graph vertices in all Voronoi cells. Therefore, by using the advanced patrolling policy, the worst-case hop distance to patrol all vertices in the entire triangulation becomes $D(V_u)|V_u| \leq n^2$ for $k = 1$, but exponential in n for LRV (Maftuleac et al., 2015). Figures 17a and 17b show simulation results of patrolling with the advanced and LRV policy for $k \geq 1$.

6.3. Centers of non-convex Voronoi cells

There are many robotic approaches to configuration control that use some variant of the weighted centroid of a region. Given the boundaries of the region, the centroid is straightforward to compute and guaranteed to be in the interior of any convex region. However, for non-convex regions, the centroid might not be inside, so approximations are often used, as in the work by Breitenmoser et al. (2010b) and Bhattacharya et al. (2013). For applications that require

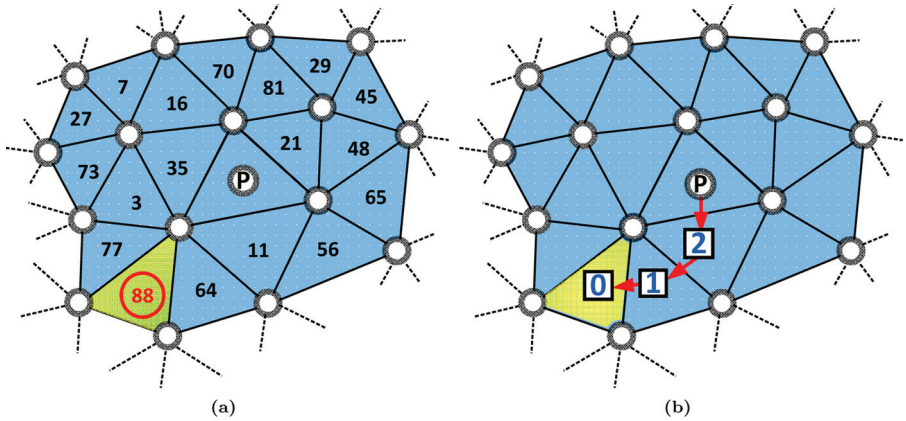


Fig. 16. Illustrations for the advanced patrolling algorithm. Blue-colored triangles indicate all triangles that belong to the Voronoi cell whose reference point is the robot P. **(a)** All triangles in the same Voronoi cell conduct a leader election algorithm to elect the maximum individual refresh time triangle (yellow colored triangle including circled number). **(b)** The yellow triangle then becomes the source of a BFS tree in the dual graph. The robot P then uses dual graph navigation to move to the source triangle. Once the robot P arrives at the triangle, it sets its individual refresh time to zero, and all triangles continuously elect a new source triangle with the maximum individual refresh time.

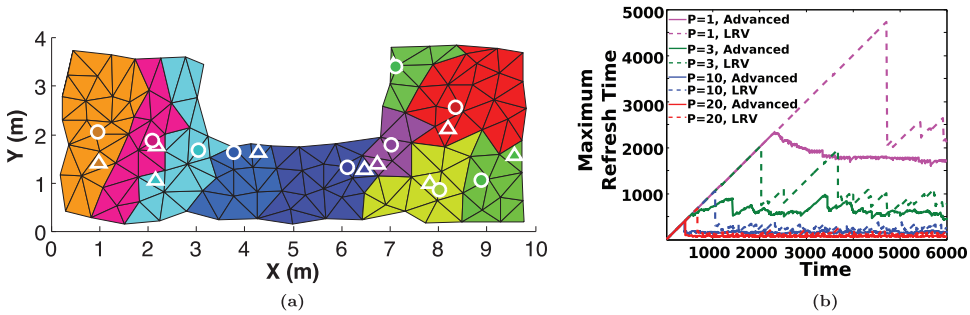


Fig. 17. **(a)** Distributed patrolling simulation screen shot (with an advanced policy). We test with 1, 3, 10, and 20 patrolling robots. Circles and equilateral triangles represent robots and the target triangle of each robot, respectively. Each color represents each topological Voronoi cell. **(b)** Comparison of the maximum refresh time. Dashed and solid lines represent the maximum refresh time result with the LRV and advanced policy using topological Voronoi cells, respectively.

navigation robots to respond to an event inside its region, the response time is determined by the geodesic distance to the event. Hence, positioning the robot with a centrality metric that uses this distance yields optimal performance. This section uses the distances in the dual graph of the triangulation to compute a *topological center* of a non-convex Voronoi cell.

There are many types of topological centrality measures (at least nine, by our count). The appropriate choice depends on the application. If we desire a fast response time for an exogeneous event in a region, the *closeness centrality* and the *eccentricity centrality* (both in the work by Koschützki et al. (2005)) minimize the average and worst-case response time, respectively.

In a distributed system, selecting a metric that uses a fixed-size message is important. In this work, we use the eccentricity centrality (EC) to provide the min-max response time, but our technique can apply to any centrality metric that can be computed using fixed-size messages. For a given vertex in the dual graph u (or its triangle, Δ) and a function to compute the shortest hop count $d_s(u, v)$ between two vertices in the dual graph, the value of the EC metric is

$$EC(\Delta) \equiv EC(u) = \frac{1}{\max_{v \in V} d_s(u, v)}$$

Given the definition of the EC metric, we then can compute the center of a cell in a brute-force fashion if *each* triangle builds a BFS tree, computes its max distance, and then selects the triangle with the min-max in the cell. The entire environment contains $|T|$ triangles, requiring $|T|$ broadcast trees, and therefore messages of size $O(|T|)$. Because $|T| \geq O(n/3)$, these are impractically sized messages.

Our solution is to compute the center iteratively, while navigating to it. Our motion controller only considers the three triangles adjacent to the one occupied by the navigating robot, triangle Δ, shown in Figure 18a. Given a metric, $m \in C_3(u)$, each triangle adjacent to Δ broadcasts a BFS tree in the dual graph, then runs a convergecast to collect

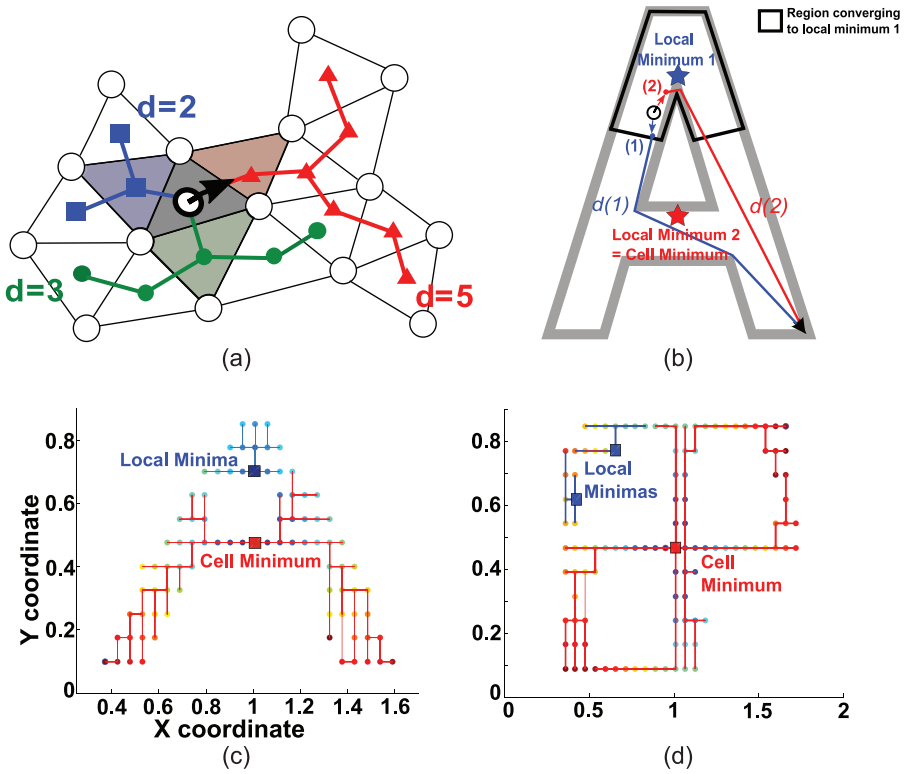


Fig. 18. (a) Distributed algorithm to choose the triangle having the min-max geodesic hops among all adjacent triangles in same topological Voronoi cell. The robot in the gray triangle builds a BFS tree rooted at each adjacent triangle and computes the min-max hop count among these three sub-trees. The min-max hop count in the red (triangles), blue (squares), and green (circles) sub-trees are 5, 2, and 3, respectively. Therefore, the robot moves into the red adjacent triangle. **(b)** An A-shaped environment where convergence to the regional local minima (blue star) happens. The agent moves toward the adjacent point at (2) because its min-max distance, $d(2)$, is less than other min-max distance, such as at $d(1)$. **(c)-(d)** Simulation results that show all converging paths from each starting vertex to the regional local minima or the regional global minimum in the dual graph of A-shaped and rectangle-shaped environments. Both of them contain at least one regional local minimum.

Algorithm 5 REALAGENTS

```
1:  while TRUE do
2:      Δ_adj ← GETADJTRIANGLES()
3:      Δ_goal ← ∅
4:      valMin ← ∞
5:      for all Δ_i ∈ Δ_adj do
6:          val ← EC(Δ_i)
7:          if val < valMin then
8:              Δ_goal ← Δ_i
9:              valMin ← val
10:         end if
11:     end for
12:     if Δ_goal ≠ ∅ then
13:         MoveRealAgentToTargetTri(Δ_goal)
14:     end if
15: end while
```

metric values. These three trees do not overlap, so the message size is $O(1)$, one broadcast and one convergecast message. The robot in Δ moves toward the adjacent triangle that has the lowest metric value. Algorithm 5 shows this iterative approach to guide a patrolling robot to the center of its topological Voronoi cell. Lines 2-9 evaluate the centrality metric on adjacent triangles. Line 13 moves the navigating robot towards the triangle with the smallest value of the metric. The process repeats until the robot is in the triangle that is the center of the cell. Unfortunately, many metrics, including EC, do not produce a unique destination with this type of local controller. Figure 18b addresses a sample environment that can generate multiple destinations and trap the navigating robots (black circle) in a regional local minimum. Figures 18c and 18d are simulation results showing that the paths of navigation robots from various dual-graph vertices converge to the regional global minimum or regional local minima in A-shaped and rectangle-shaped environments.

To address the regional local minima problem, we use a probabilistic approach. Algorithm 6 shows pseudocode for this approach. With probability p, each triangle creates a *navigation agent*. Each of these agents broadcasts a

Algorithm 6 VIRTUALAGENTS

Initially, each owner that has the physical data structure of its triangle randomly generates a message which stands for a virtual agent in the triangle with probability, p. An owner robot of triangle T_i then runs the below algorithm.
while ISREALAGENTINCENTROID() = *FALSE* **do**
  $T_{adj} \leftarrow$ GETADJTRIANGLES()
  $maxHop \leftarrow \infty$
  $T_{max} \leftarrow \emptyset$
  for all $T_j \in \mathcal{T}_{adj}$ **do**
    $h_j \leftarrow$ GETMINMAXHOP(T_j)
    if $h_j < maxHop$ **then**
      $T_{max} \leftarrow T_j$
      $maxHop \leftarrow h_j$
    end if
  end for
  if $T_{max} = \emptyset$ **then**
    ELECTCELLMIN()
    if *IsCellMin*(T_i) **then**
      GuideRealRobotToTri(T_i)
    end if
  else
    M*oveVirtualAgentToTargetTri*(T_{max})
  end if
end while

message throughout the cell of the form {agentID, agentMetricValue, messageHops}. We allocate k slots for broadcast messages, so a triangle only creates an navigation agent if there is a free slot. The navigation agents follow the same motion policy as the navigation robots, but propagate at network speeds. The physical robot follows the broadcast message from the agent with the best *agentMetricValue*. If multiple agents reach the same triangle, one remains; the others are destroyed. An agent that remains stationary for more than $2D(\mathcal{V}_u)$ rounds without having the best metric value in $\mathcal{V}_u$ is destroyed. This always leaves one agent present – the one that is in the triangle with the best metric value found. This process continues indefinitely, with one agent remaining at the current best value, and $k-1$ agents in motion. Eventually, a virtual agent will be created in the basin of the cell's regional global minimum, will propagate there, and will remain indefinitely. Increasing the number of agents reduces the time required to find the best value in $\mathcal{V}_u$, but adds communications cost. Figure 19 illustrates how this method guides an actual robot to a region's global minimum with virtual agents.

Each agent has the same communications requirements as the navigating robot: a broadcast+convergecast message to find the adjacent triangle with the deepest tree in the cell. Therefore the total communications complexity for each cell is $O(k+1)$. Note that these messages do not travel outside of the cell, so the total global message complexity for the entire system is also $O(k+1)$, and it does not depend on

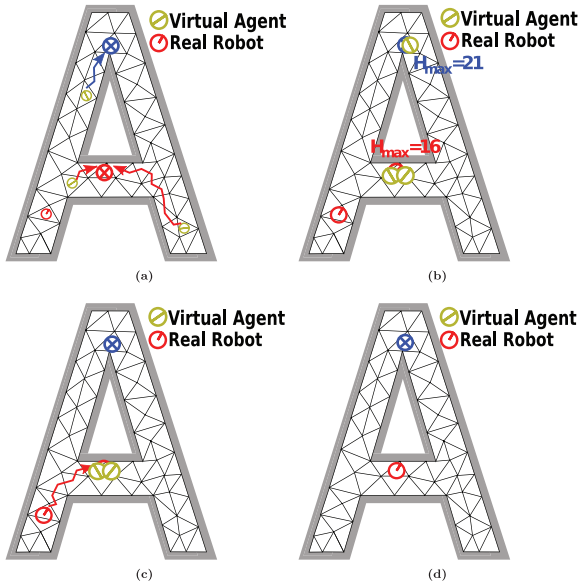


Fig. 19. Illustrations of guiding a robot to the regional minimum in an 'A'-shaped Voronoi cell using virtual agents. Blue and red circles with 'X' marks denote regional local minima and the regional minimum, respectively. **(a)** Each cell generates k virtual agents. Each virtual agent continuously moves to the min-max hop count adjacent triangle among all adjacent triangles, and broadcasts the min-max hop count of the current triangle (containing the virtual agent to elect the regional min-max hop count among all virtual agents in the cell). At the same time, the triangle with the elected regional minimum becomes a source to guide a real robot to the triangle. Note that this source triangle may be changed as any virtual agent moves. **(b)** All agents eventually converge to either the regional local minima or regional minimum, and then the elected source triangle and the regional minimum among min-max hops of all virtual agents are changed no more. The final source triangle in which the real robot will converge becomes the regional minimum triangle with non-zero probability, and builds its own dual graph in the cell to guide the real robot. **(c)-(d)** The real robot then uses dual graph navigation to move to the agent with the regional global minimum.

the number of cells (i.e. the number of robots). The running time requires worst-case $O(k \cdot D(G^D))$ rounds to select the initial k agents. It takes $O(D(G^D))$ rounds for an agent to travel the entire region, and $O(D(G^D))$ rounds for a new agent to be created when two reach the same triangle.

This algorithm is probabilistically complete: if we conduct an infinite number of trials, the probability that at least one agent is created in the basin of the region's global minimum, and therefore converges towards it, becomes 1. However, the expected running time will depend on the number of navigation agents deployed – generating more agents will find the regional global minimum faster. This lets us trade off between message size and expected execution time. For simplicity, we assume that a set of k navigation agents is generated, which then converge (or are destroyed) once every $D(G^D)$ rounds. Therefore, the expected time to start

an agent in the basin of the cell's regional global minimum is governed by a series of Bernoulli trials, where each trial takes place every $D(G^D)$ rounds. Note that actual navigation agents are regenerated right after they converge (or are destroyed), so that the life span of each agent may be shorter than $D(G^D)$ rounds. Therefore, the expected execution time of these Bernoulli trials is actually an upper bound for the actual expected execution time.

Let F ("fail") denote the event that all k virtual agents are created in triangles that are in the basin of a regional local minimum within V_u. We define $|V_u|$ as the total number of triangles in region V_u, and $|V_u|_{\text{local}}$ as the the number of triangles in V_u that converge to a regional local minimum. The probability of event F is given by

$$P(F) = \frac{|V_u|_{\text{local}}(|V_u|_{\text{local}} - 1)...(|V_u|_{\text{local}} - k + 1)}{|V_u|(|V_u| - 1)...(|V_u| - k + 1)}$$

We can therefore define "success", S, as an event that at least one of k agents successfully converges to the regional global minimum. The probability of the event S, $P(S)$ is 1-$P(F)$.

We then consider the geometric probability in a negative binomial experiment, where $P(1) = P(S)$ and $P(0) = P(F)$. The geometric probability of i_th trial is $P(S)P(F)^{i-1}$. Therefore, the expected trial number until the first appearance of event S, $E_N(S)$, is given by

$$E_N(S) = \sum_{i=0}^{\infty} iP(S)P(F)^{i-1} = \sum_{i=0}^{\infty} i(1 - P(F))P(F)^{i-1}$$
$$= \sum_{i=0}^{\infty} P(F)^i = \frac{1}{1 - P(F)} = \frac{1}{P(S)}$$

The maximum lifespan of an agent in one trial is $D(G^D)$ rounds, so the expected execution time, $E(S)$, becomes

$$E(S) = E_N(S)D(G^D) = \frac{D(G^D)}{P(S)}$$

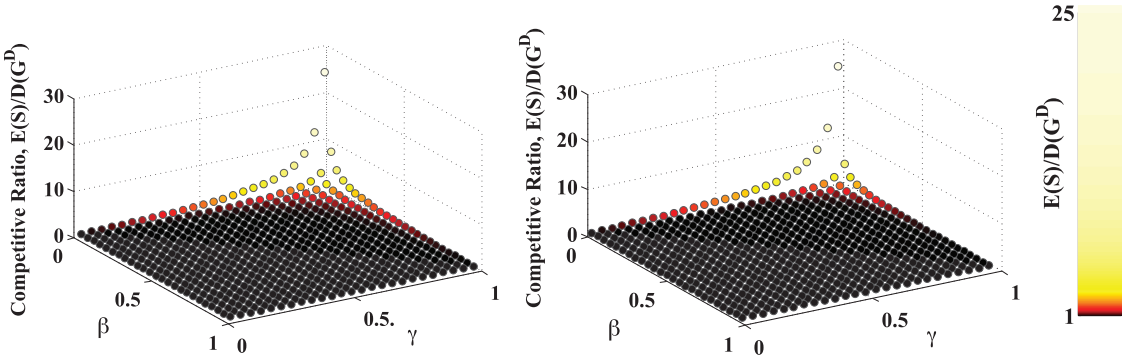
With this, we can show the relationship between communications bandwidth (the number of agents, k) and execution time. In the extreme case, if each triangle creates an agent during each trial, the number of navigation agents is $|V_u|$, and $P(F) = \frac{|V_u|_{\text{local}}(|V_u|_{\text{local}}-1)\times...\mathbf{0}...\times(|V_u|_{\text{local}}-|V_u|+1))}{1+2+...+|V_u|} = \mathbf{0}$ and the expected execution time becomes $D(G^D)$, but the message size is $|V_u|$. As we reduce the number of navigation agents, $P(F)$ increases and the expected execution time also increases by $\frac{1}{1-P(F)}D(G^D)$.

Fortunately, it is possible to achieve a reasonable expected execution time with a relatively small number of navigation agents. Figures 20a–20b show simulation results to illustrate the relationships between the expected execution time, the ratio of the number of agents k to $|V_u|$ (β), and the ratio of $\frac{|V_u|_{\text{local}}}{|V_u|}$ (γ). Even in extreme environments whose $|V_u|_{\text{local}}$ values are 90% of $|V_u|$ (meaning that 90% of triangles converge to regional local minima), the expected execution time of all cases were below $2D(G^D)$ using six agents, or messages of size $0.12|V_u|$. Even with one virtual agent, (k=1), the expected execution time is less than $2D(G^D)$, as long as the ratio of $\frac{|V_u|_{\text{local}}}{|V_u|}$ is less than 0.5, which is true for many environments, including the 21 studied in this paper.

To test navigation to the topological center, we generated 25 cases of $(3.81, 0.31$ rad)-*fat* triangulations. Figure 21 shows these triangulations with two robots (in red), which create two topological Voronoi cells. Each cell uses two virtual agents. As the robots move to optimal locations, the value of the *EC* metric converges to the same value for both robots. We compared topological tessellation with and without navigation agents. Figure 22 shows the results of six trials, compared to a high-resolution discretization of the environment to approximate the continuous solution. The trials without navigation agents often got stuck in regional local minima, producing a high variance and larger mean. The navigation agents nearly eliminated this problem, producing results almost as good as the ideal solution. These results are difficult to directly compare to existing work. The closest work, Pasqualetti et al. (2012b), uses global knowledge of the environment geometry, global localization, long-range communications, and square tessellations of the workspace. Our approach uses a triangular tessellation, is fully distributed, uses local communications, and has no shared global knowledge. However, in the limit of arbitrarily small tessellation elements, our two approaches will both converge to the geodesic center.

Finally, let us recall the message size of the brute-force algorithm stated in Section 6.3. The problem of the brute force method lies in the size of message. Suppose that we have n low-cost robots for triangulation, p actual robots acting like sites in each topological Voronoi cell, k virtual agents in each topological Voronoi cell, and $|T|$ triangles made by the n low-cost robots.

Let us move to the comparison of message size between the brute-force and the virtual agent method. For simplicity, we assume that $p = 1$, so that there is only one Voronoi cell in a triangulated space. First, the brute-force algorithm computes the maximum hop count from all triangles and elects the minimum hop count among them to compute the center. In other words, all $|T|$ triangles in a triangulation become source triangles and build $|T|$ BFS trees to compute the maximum hop count from each source triangle. While all triangle owners construct $|T|$ BFS trees simultaneously, each robot keeps receiving and broadcasting a message containing $|T|$ distinctive hop values from $|T|$ source triangles. Therefore, the size of a message used for the brute-force method becomes at least $|T| \times 8$ bits (uint8, 16 bits for uint16 or 32 bits for uint32) = $O(|T|)$. Note that the size of a message is proportional to the number of triangles (or the number of owner robots). This reduces the advantage of a distributed approach, including algorithm scalability; if the population of robots increases, the size of message also



(a) $|V_u| = 50$ (b) $|V_u| = 100$

Fig. 20. Expected execution times given number of $|V_u|$. β and γ indicate the ratio $\frac{k}{|V_u|}$ and $\frac{|V_u|_{\text{local}}}{|V_u|}$, respectively. In all cases, just a small increase in β or small decrease in γ results in a huge reduction of the competitive ratio of the expected execution time (i.e. the expected execution time rapidly converges to the $O(D(G^D))$ rapidly).

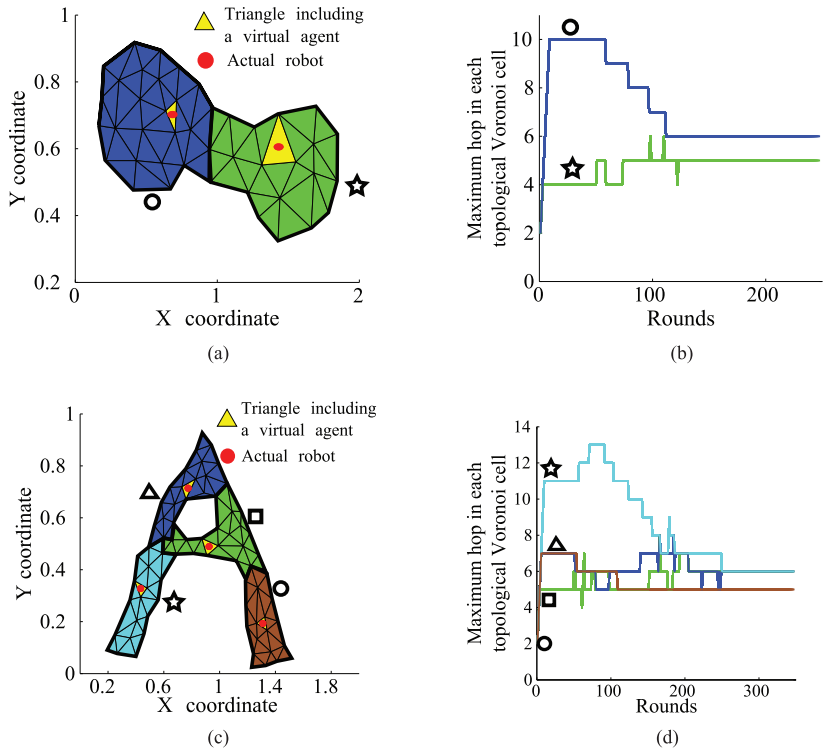


Fig. 21. Resulting Voronoi tessellation in a dumbbell-shaped **(a)** and A-shaped environment **(c)**. Yellow triangles and red dots indicate triangles containing virtual agents and actual robots, respectively. Maximum hops of all topological Voronoi cells at each round in a dumbbell-shaped **(b)** and A-shaped **(d)** environment. As actual robots move toward the desired centroids, the maximum hop graphs of each cell converge to the similar value with each other.

becomes impractical and a robot has to broadcast a large message within limited communication bandwidth.

On the other hand, the virtual agent method requires constant-size messages. Each message contains p variables to select which Voronoi cell each triangle should belong to and k variables to compute the minimum max-hop count virtual agent by building k BFS trees from each virtual agent in a Voronoi cell. Therefore, the size of a single message in the virtual agent method becomes $O(p + k)$. Note that this work considers a scenario in which a large number of low-cost robots supports a few more powerful robots. Therefore, p should be much smaller than $|T|$ or n ($n > 3|T| >> p$). For the k variable, however, there is a trade-off between the message size and algorithm performance; if we use a large size message with large k value, the expected execution time, $E(S)$, decreases. Fortunately, it turns out that a relatively small message size k is enough to achieve a good $E(S)$, as shown in Figure 20. For instance, in Figure 20b, $E(S)$ for $k = 5$ (i.e. $E(S)_{k=5}$) is only 10.7% larger than $E(S)_{k=100}$. The brute-force method

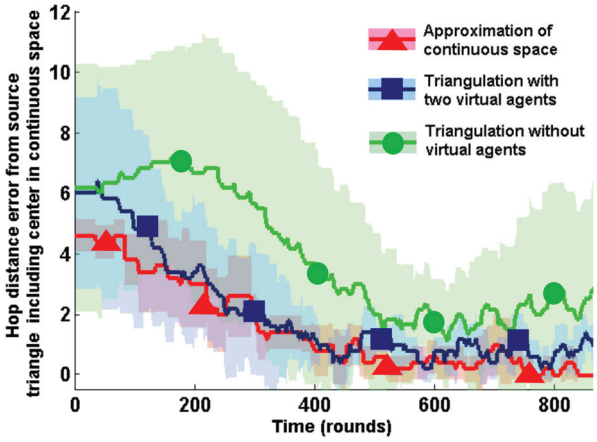


Fig. 22. A plot that shows an error distance (in hop) between the triangle containing the geodesic center in continuous space and the triangle containing an actual robot at every round in *A*-shaped environment. Red, blue, and green lines indicate the hop count error with two virtual agents in approximation of continuous space, two virtual agents in triangulation, and no virtual agents in triangulation, respectively.

for $E(S)_{k=100}$ requires 25 times larger messages ($|T|$=100, if p=1). In summary, the virtual agent algorithm significantly reduces the required message size ($O(|T|) >> O(k+p)$). In addition, k and p are user definable and are not related to the number of robots, n. Therefore, the virtual agent method preserves algorithm scalability of a distributed method.

7. Conclusion

This paper presents distributed algorithms for the construction of a triangulation using a multi-robot system and its applications. Instead of enhancing a single robot to be more powerful, we make the environment smarter by deploying many low-cost robots in a structured way, to form a triangulated robot network. Data stored in the triangles is embedded in the physical world, creating a *physical data structure*.

The triangulation maps useful geometric features of its surrounding environment into topological features, notably to the dual graph of adjacent triangles. The physical data structure allows a set of robots with limited capability accomplish complex tasks, including the computation of Voronoi tessellation and patrolling, without powerful sensors or a geometric map of the environment.

The key feature of our work is that it combines methods from robotics, sensor networks, and algorithmics, providing an analytic and experimental framework for further, high-level approaches. In an ongoing project, we investigate distributed topology extraction by using the triangles to identify corridors, junctions, and dead-ends in a maze-like environment. One of the tasks involved is to fully understand and control the relationship between the quality of a triangulation and the accuracy of extracted topology. Another challenge is to extend the aspects of robustness: while our algorithms that use the triangulation are robust to robot failures and communications errors, the triangulation itself is not self-stabilizing. Thus, rebuilding the physical data structure after robot failures and movements deserves further investigation; we are considering a variety of different approaches for this.

Acknowledgments

Preliminary extended abstracts summarizing parts of this work have appeared in the short conference papers Lee et al. (2014a), Lee et al. (2014b). We thank several anonymous referees for various helpful comments that helped to improve the presentation of this paper.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Science Foundation under CPS-1035716 and CNS-1330085.

References

Alpern S and Gal S (2003) *The theory of search games and rendezvous*. International Series in Operations Research and Management Science. Boston: Kluwer Academic Publishers.

Aronov B (1989) On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica* 4(1-4): 109–140.

Batalin M and Sukhatme GS (2004) Using a sensor network for distributed Multi-Robot task allocation. In: *IEEE International Conference on Robotics and Automation*, New Orleans, USA, 26 April–1 May 2004, pp. 158–164.

Becker A, Demaine ED, Fekete SP et al. (2013a) Reconfiguring massive particle swarms with limited, global control. In: *Algorithms for Sensor Systems - 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, Sophia-Antipolis, France, 5–6 September 2013, pp. 51–66.

Becker A, Demaine ED, Fekete SP et al. (2014) Particle computation: Designing worlds to control robot swarms with only global signals. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, 31 May–7 June 2014, pp. 6751–6756.

Becker A, Demaine ED, Fekete SP et al. (2015) Tilt: The video - designing worlds to control robot swarms with only global signals. In: *31st International Symposium on Computational Geometry (SoCG)*, Eindhoven, Netherlands, 22–25 June 2015, pp. 16–18.

Becker A, Fekete SP, Kröller A et al. (2013b) Triangulating unknown environments using robot swarms. In: *29th*

Annual ACM Symposium Computational Geometry (SoCG). Rio de Janeiro, Brazil, 17–20 June 2013, pp. 345–346. Video available at http://imaginary.org/film/triangulating-unknown-environments-using-robot-swarms.

Bern M and Eppstein D (1992) Mesh generation and optimal triangulation. In: Du DZ and Hwang FK (eds) *Computing in Euclidean Geometry*, *Lecture Notes Series on Computing volume 1*. Singapore: World Scientific, pp. 23–90.

Bhattacharya S, Michael N and Kumar V (2013) Distributed coverage and exploration in unknown non-convex environments. In: *Distributed Autonomous Robotic Systems*. Lausanne, Switzerland: Springer, pp. 61–75.

Bredin J, Demaine E, Hajiaghayi M et al. (2010) Deploying sensor networks with guaranteed fault tolerance. *IEEE/ACM Transactions on Networking* 18(1): 216–228.

Breitenmoser A, Metzger JC, Siegwart R et al. (2010a) Distributed coverage control on surfaces in 3D space. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 18–22 October 2010, pp. 5569–5576.

Breitenmoser A, Schwager M, Metzger JC et al. (2010b) Voronoi coverage of non-convex environments with a group of networked robots. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, 3–8 May 2010, pp. 4982–4989.

Burgard W, Moors M, Fox D et al. (2000) Collaborative multi-robot exploration. In: *IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, USA, 24–28 April 2000, pp. 476–481.

Caicedo-Nuez C and Zefran M (2008) A coverage algorithm for a class of non-convex regions. In: *47th IEEE Conference on Decision and Control (CDC)*, Cancun, Mexico, 9–11 December 2008, pp. 4244–4249.

Cortes J, Martinez S, Karatas T et al. (2004) Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation* 20(2): 243–255.

Durham JW, Carli R, Frasca P et al. (2012) Discrete partitioning and coverage control for gossiping robots. *IEEE Transactions on Robotics* 28(2): 364–378.

Durham JW, Franchi A and Bullo F (2010) Distributed pursuit-evasion with limited-visibility sensors via frontier-based exploration. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, USA, 3–8 May 2010, pp. 3562–3568.

Efrat A, Fekete SP, Gaddehosur PR et al. (2008) Improved approximation algorithms for relay placement. In: *16th Annual European Symposium on Algorithms (ESA)*, Karlsruhe, Germany, 15–17 September 2008, pp. 356–367.

Elmaliach Y, Agmon N and Kaminka GA (2009) Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence* 57(3-4): 293–320.

Fekete SP, Kamphans T, Kröller A et al. (2011) Exploring and triangulating a region by a swarm of robots. In: *14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 6845 of *LNCS*, Princeton, NJ, USA, 17–19 August 2011, pp. 206–217.

Fekete SP, Kaufmann M, Kröller A et al. (2005) A new approach for boundary recognition in geometric sensor networks. In: *17th Canadian Conference on Computational Geometry*, Windsor, ON, Canada, 10–12 August 2005, pp. 82–85. Windsor, ON, Canada: University of Windsor.

Fekete SP and Kröller A (2006) Geometry-based reasoning for a large sensor network. In: *22nd Annual ACM Symposium on Computational Geometry (SoCG)*, Sedona, AZ, USA, 5–7 June 2006, pp. 475–476.

Fekete SP and Kröller A (2007) Topology and routing in sensor networks. In: *3rd International Workshop on Algorithmic Aspects Wireless Sensor Networks (ALGOSENSORS)*, Wroclaw, Poland, 14 July 2007, pp. 6–15.

Fekete SP, Kröller A, Pfisterer D et al.(2004) Neighborhood-based topology recognition in sensor networks. In: *Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, Turku, Finland, 16 July 2004, pp. 123–136.

Fekete SP, Rex S and Schmidt C (2013) Online exploration and triangulation in orthogonal polygonal regions. In: *7th International Workshop on Algorithms and Computation (WALCOM)*, volume 7748 of *LNCS*, Kharagpur, India, 14–16 February 2013, pp. 29–40.

Fischer MJ, Lynch NA and Paterson MS (1985) Impossibility of distributed consensus with one faulty process. *Journal of the ACM* 32(2): 374–382.

Geraerts R (2010) Planning short paths with clearance using explicit corridors. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, USA 3–8 May 2010, pp. 1997–2004.

Ghoshal A and Shell DA (2011) Being there, being the RRT: Space-filling and searching in place with minimalist robots. In: *AAAI Spring Symposium: Multirobot Systems and Physical Data Structures*, Palo Alto, CA, USA, 21–23 March 2011.

Hsiang TR, Arkin EM, Bender MA et al. (2004) Algorithms for rapidly dispersing robot swarms in unknown environments. In: *International Workshop on Algorithmic Foundations of Robotics (WAFR)*, volume 7 of *STAR*, Utrecht/Zeist, Netherlands, 11–13 July 2004, pp. 77–93.

Kallmann M (2005) Path planning in triangulations. In: *Proceedings of the IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, Edinburgh, Scotland, 31 July 2005, pp. 49–54.

Kashyap A, Khuller S and Shayman M (2011) Relay placement for fault tolerance in wireless networks in higher dimensions. *Computational Geometry* 44(4): 206–215.

Ko J, Stewart B, Fox D et al. (2003) A practical, decision-theoretic approach to multi-robot mapping and exploration. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 4, Las Vegas, NV, USA, 27 October–1 November 2003, pp. 3232–3238.

Konolige K, Fox D, Ortiz C et al. (2006) Centibots: Very large scale distributed robotic teams. In: *Experimental Robotics IX*. Singapore: Springer, pp. 131–140.

Koschützki D, Lehmann KA, Peeters L et al. (2005) Centrality indices. In: *Network Analysis*. Springer, pp. 16–61.

Kröller A, Fekete SP, Pfisterer D et al. (2006) Deterministic boundary recognition and topology extraction for large sensor networks. In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Miami, FL, USA, 22–24 January 2006, pp. 1000–1009.

Kröller A, Pfisterer D, Fekete SP et al. (2009) Algorithms and simulation methods for topology-aware sensor networks. In: Lerner J, Wagner D and Zweig KA (eds) *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*. Springer, pp. 380–400.

Krupke D, Ernestus M, Hemmer M et al. (2015) Distributed cohesive control for robot swarms: Maintaining good connectivity in the presence of exterior forces. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Gemany, 28 September–3 October 2015, pp. 413–420.

Lee SK, Becker A, Fekete SP et al. (2014a) Exploration via structured triangulation by a multi-robot system with bearing-only low-resolution sensors. In: *IEEE International Conference on Robotics and Automation (IROS)*, Hong Kong, China, 31 May–7 June 2014, pp. 2150–2157.

Lee SK, Fekete SP and McLurkin J (2014b) Geodesic topological voronoi tessellations in triangulated environments with multi-robot systems. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Chicago, IL, USA, 14–18 September 2014, pp. 3858–3865.

Li Q and Rus D (2005) Navigation protocols in sensor networks. *ACM Transactions on Sensor Networks* 1(1): 3–35.

McLurkin J (2008) *Analysis and implementation of distributed algorithms for multi-robot systems*. PhD thesis, Massachusetts Institute of Technology, USA.

McLurkin J and Demaine ED (2009) A distributed boundary detection algorithm for multi-robot systems. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA, 11–15 October 2009, pp. 4791–4798.

McLurkin J, Rykowski J, John M et al. (2013) Using multi-robot systems for engineering education: teaching and outreach with large numbers of an advanced, low-cost robot. *IEEE Transactions on Education* 56(1): 24–33.

McLurkin J and Smith J (2004) Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In: *Proc. 7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, Toulouse, France, 23–25 June 2004, pp. 399–408.

Maftuleac D, Lee SK, Fekete SP et al. (2015) Local policies for efficiently patrolling a triangulated region by a robot swarm. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, 26–30 May 2015, pp. 1809–1815.

Nouyan S and Dorigo M (2006) Chain based path formation in swarms of robots. In: *Ant Colony Optimization and Swarm Intelligence*. Brussels, Belgium: Springer, pp. 120–131.

O'Hara KJ, Walker DB and Balch TR (2005) The GNATs - low-cost embedded networks for supporting mobile robots. In: *Multi-Robot Systems. From Swarms to Intelligent Automata, Volume III*. Springer, pp. 277–282.

Olson E (2011) Apriltag: A robust and flexible visual fiducial system. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 9–13 May 2011, pp. 3400–3407.

Pasqualetti F, Durham JW and Bullo F (2012a) Cooperative patrolling via weighted tours: Performance analysis and distributed algorithms. *IEEE Transactions on Robotics* 28(5): 1181–1188.

Pasqualetti F, Franchi A and Bullo F (2012b) On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms. *IEEE Transactions on Robotics* 28(3): 592–606.

Pimenta L, Kumar V, Mesquita RC et al. (2008) Sensing and coverage for a network of heterogeneous robots. In: *47th IEEE Conference on Decision and Control (CDC)*, Cancun, Mexico, 9–11 December 2008, pp. 3947–3952.

Rubenstein M, Cornejo A and Nagpal R (2014) Programmable self-assembly in a thousand-robot swarm. *Science* 345(6198): 795–799.

Rutishauser S, Correll N and Martinoli A (2009) Collaborative coverage using a swarm of networked miniature robots. *Robotics and Autonomous Systems* 57(5): 517–525.

Spears WM, Spears DF, Hamann JC et al. (2004) Distributed, physics-based control of swarms of vehicles. *Autonomous Robots* 17(2): 137–162.

Suri S, Vicari and Widmayer P (2008) Simple robots with minimal sensing: From local visibility to global geometry. *International Journal of Robotics Research* 27: 1055–1067.

Turgut AE, Çelikkanat H, Gökçe F et al. (2008) Self-organized flocking in mobile robot swarms. *Swarm Intelligence* 2(2–4): 97–120.

Yun SK and Rusy D (2012) Distributed coverage with mobile robots on a graph: Locational optimization. In: *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, USA, 14–18 May 2012, pp. 634–641. IEEE.