2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Exploration via Structured Triangulation by a Multi-Robot System with Bearing-Only Low-Resolution Sensors

Seoung Kyou Lee, Aaron Becker, Sándor P. Fekete, Alexander Kröller, and James McLurkin

***Abstract*— This paper presents a distributed approach for exploring and triangulating an unknown region using a multi-robot system. The resulting triangulation is a *physical data structure* that is a: compact representation of the workspace, contains distributed knowledge of each triangle, builds the dual graph of the triangulation, and supports reads and writes of auxiliary data. Our algorithm builds a triangulation in a closed two-dimensional Euclidean environment, starting from a single location. It provides coverage with a breadth-first search pattern and completeness guarantees. We show that the computational and communication requirements to build and maintain the triangulation and its dual graph are small. We then present a physical navigation algorithm that uses the dual graph, and show that the resulting path lengths are within a constant factor of the shortest-path Euclidean distance. Finally, we validate our theoretical results with experiments on triangulating a region with a system of low-cost robots. Analysis of the resulting triangulation shows that most of the triangles are of high quality, and cover a large area. Implementation of the triangulation, dual graph, and navigation all use communication messages of fixed size, and are a practical solution for large populations of low-cost robots.**

I. Introduction and Related Work

Many practical applications of multi-robot systems, such as search-and-rescue, exploration, mapping and surveillance require robots to disperse across a large geographic area. Large populations of robots offer two large advantages: they can search the environment rapidly using a breadth-first approach, and can maintain coverage of the environment after the dispersion is complete.

In this paper, we demonstrate that triangulating the workspace with a multi-robot system is a useful approach to dispersion and monitoring. Triangulations are used in a large variety of applications because of their useful properties. In our application they provide complete coverage, they can be built with only basic local geometry, and they allow proofs of properties for coverage, navigation, and distributed data storage. The underlying topological structure of a triangulation allows us to exploit its dual graph for mapping and routing, with performance guarantees for these purposes. Fig. 1 shows an example output demonstrating a triangulated network, its dual graph, and a navigating robot.

We are interested in solutions for large populations of robots, and focus our attention on approaches applicable on small, low-cost robots with limited sensors and capabilities. In this work, we assume that robots do not have a map of the environment, nor the ability to localize themselves relative to the environment geometry, *i.e.* SLAM-style mapping is beyond the capabilities of our platform. We exclude solutions that use centralized control, as the communication and processing constraints do not allow these approaches to scale to large populations. We also do not assume that GPS localization or external communication infrastructure is available, which are limitations present in an unknown indoor environment. Finally, we assume that the communication range is much smaller than the size of the environment, so a multi-hop network is required for communication, and the *local network geometry* provides each robot with geometric information about its neighboring robots.

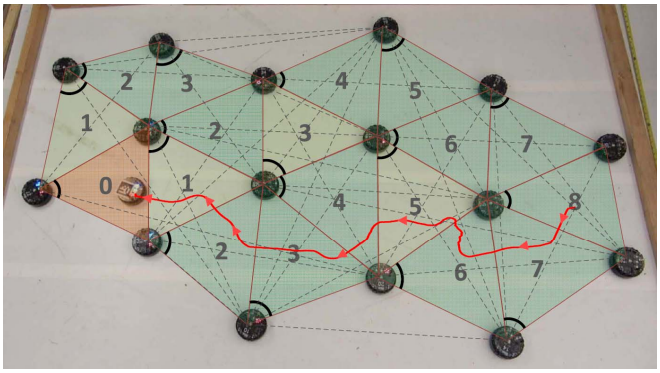


Fig. 1: A sample triangulation and navigation experiment result with 17 r-one robots. The planar network of triangulation is a subset of the full network (dashed gray lines). Except two right most robots, each robot creates a new triangle (dark green) by expanding toward the frontier and may discover new triangles (light green triangles) by examining local network geometry. Small black arcs indicate which robot creates and stores which triangle. This is a distributed *physical data structure*; there is no centralized storage of triangulation information. The network between adjacent triangles forms a *dual graph* of the triangulation. In this example, a navigating robot used a tree rooted at the red triangle to guide the navigation robot from its current location to the goal location, and followed the red path. The numbers indicate hops in the dual graph from the goal triangle.

The basic problem requires exploring an unknown region by triangulation from a given starting position. The maximum edge length of a triangle is bounded by the communications range of the robots. 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). Both problems have been studied both

S. Lee, A. Becker, and J. McLurkin are with the Computer Science Department, Rice University, Houston, TX, 77005 USA e-mail: sl28@rice.edu.

A. Kröller and S. Fekete are with the Computer Science Department, TU Braunschweig, Braunschweig, Germany.

for the *offline* scenario, in which the region is fully known, and the *online* scenario, where the region is not known in advance [1]. 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. In this work we focus on the online MATP problem. Our algorithm extends the covered region by adding new triangles to the frontier of the exploration. The motion controllers we present use local geometric information. In particular, we focus on a simple platform that can only measure angles between neighbors and detect nearby obstacles. We provide a number of results:

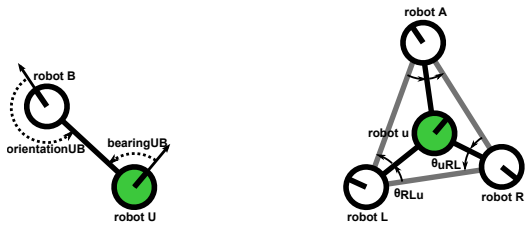
- We develop simple and efficient exploration methods based on triangulation.
- We show that these methods only require local information and geometry.
- We demonstrate that well-known abstract concepts (such as the dual graph) can be implemented in a distributed network of robots.
- We provide provable performance guarantees for online triangulation and routing.
- We demonstrate the practicality of our method by implementing it with simple, low-cost robots. (See our video [2] for an overview of [1].)

Related Work

Our work combines ideas of routing in stationary sensor networks [3] with approaches for dynamic robot swarms. New challenges arise from considering a large number of real-life mobile nodes with limited capabilities. Classical triangulation problems seek a triangulation of all vertices of a polygon, but allow arbitrary edge length [1]. This differs from our model, in which edge lengths are bounded by communication range. 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 [4].

The problem of placing a minimum number of relays with limited communication range in order to achieve a connected network (a generalization of the classical Steiner tree problem) was considered by Efrat et al. [5], who gave 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 was considered by Bredin et al. [6], who asked for the minimum number of relays to be placed to assure a k-connected network. They presented approximation results for the offline problem. For swarms, Hsiang et al. [7] 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.

This work builds on previous results in multi-robot exploration. McLurkin and Smith, and Konolige et al. [8], [9] presented a breadth-first distribution from a more practical view, using a swarm of 100 robots. Durham et al. [10] presented a pursuit-evasion algorithm for a team of robots, and Ghoshal and Shell [11] describe an RRT-like algorithm. Both use the robot's physical positions as a data structure to store intermediate results of the algorithm. Spears and Spears described algorithms to produce a triangle lattice, but this is not a *triangulation*: there is no knowledge of triangles, the dual graph, or distributed data structures for computation [12].



(a) Bearing and orientation definitions. **(b)** Measuring triangle angles using 2-hop coordinate system.

Fig. 2: (a) Robot u can measure the bearing to neighbor u_B, $B_u(u_B)$, and the orientation of neighbor u_B, $Ori(u_B)$. **(b)** triangle angles (black arrows) are measured from neighbors of robot u, and shared with u using a local broadcast message. We define the left (right) inner angle for the triangle angle of u's left (right).

II. Model and Assumptions

We have a system of n 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 $N(u) = \{v \in V \mid \{u, v\} \in E\}$. Robot u sits at the origin of its local coordinate system, with the $\hat{x}$-axis aligned with its current heading. Each robot can measure the angles of the geometry of its local network, as shown in Fig. 2a. Robot u cannot measure distance to its neighbors, but can only measure the *bearing* and *orientation*. We assume that these angular measurements have limited resolution.

Robots share their angle measurements with their neighbors so robot u learns all angles in its 2-hop neighborhood. Fig. 2b shows the *inner angles* of a triangle around u. These angles are measured by u's neighbors, then broadcast to u. The communication used is $O(\max(\delta(u) \in V)^2)$ messages/robot/round, $\delta(u)$ is the degree of vertex u.

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 ≈ 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 [13]. 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

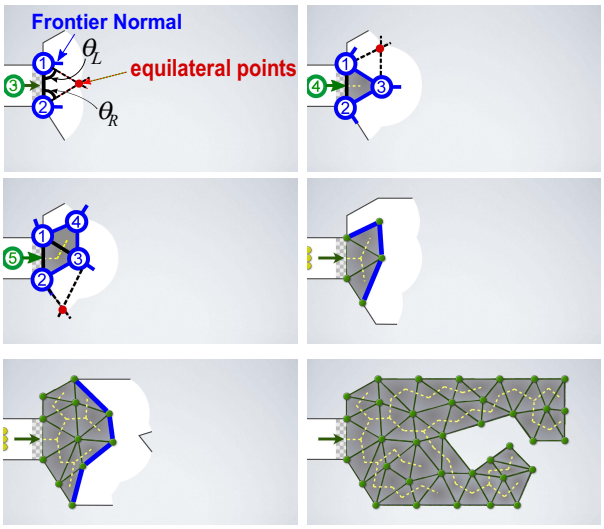


Fig. 3: Constructing a triangulation in a BFS manner. The frontier edges are blue and interior edges are green. The edges of the BFS tree (the dual graph) are yellow. Each robot creates a new triangle when it arrives at the *equilateral point*. The blue tick marks on each robot show the direction of the *frontier normal*. This points into unexplored space, in the direction perpendicular to the frontier edges incident at each robot.

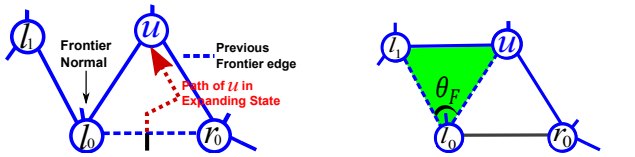
$v \in N(u)$. Each message contains a set of public variables, including the sending robot's unique ID number $v.id$. The remaining variables will be defined later, but we note that the number of bits needed for each variable is bounded by $\log_2 n$, i.e. the number of bits required to identify each robot. This produces a total message of constant size.

III. Max-Area Triangulation Algorithm

Fig. 3 illustrates the execution of the Max-Area Triangulation (MAT) algorithm. Initially, two *base robots* mark the *base edge*—such as a door to an unexplored building. 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 Fig. 3, 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.

During algorithm execution, we distinguish the following types of edges in the robot network G: **1)** *Frontier edges* (blue lines in Fig. 3), $\{u, v\} \in E_F$, which belong to only one triangle and have at least one vertex that is not in contact with the wall. **2)** *Internal edges*, $\{u, v\} \in E_I$ which belong to two adjacent triangles. **3)** *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. The yellow lines indicate the dual graph, D, which connects adjacent triangles. We will address details of the dual graph in Section III-B.



(a) Forming an *Expansion* triangle **(b)** Forming a *Discovery* triangle

Fig. 4: (a) An example *Expansion* triangle. Red arrow shows the path of u in *Expanding* state. After arriving at the equilateral point, u switches to *Expanded* state. **(b)** An example *Discovery* triangle (light green). u checks θ_F to evaluate the quality of candidate triange, $\Delta u l_1 l_0$.

A. Triangulation

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, these algorithms are described in Sections III-B and III-C.

A new triangle can be formed in two ways, *expansion* or *discovery*. Fig. 4a 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 Fig. 4a). 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. Because the edge $\{l, r\}$ is now internal, it is not used for expansion again, which prevents creating overlapping triangles.

When u enters the expanded state, it needs to discover all of the unexpanded high-quality triangles adjacent to Δulr. Fig. 4b 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, $\Delta u l_i l_{i-1}$ (light green in Fig. 4b), and evaluates its quality using definition 3.1. The search terminates if the triangle is high-quality, or if there are no further neighbors to consider. 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.

B. Dual Graph Construction

The dual graph of our a triangulation, D, describes the adjacencies between adjacent triangles. The dual graph can be used for realizing global objectives, such as routing. However, one difficulty for a distributed swarm of robots is the absence of a 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 to edges between robots in the primal graph.

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 two triangle owners connected by an edge in the dual graph can communicate with each other through the primal graph. This is trivial for two triangles Δ_1 and Δ_2 owned by the same robot, so we must show that for two different triangle owners $o(\Delta_1) \neq o(\Delta_2)$ with a dual graph edge, $\{o(\Delta_1), o(\Delta_2)\}_D \in D$, $\{o(\Delta_1), o(\Delta_2)\}$ is an edge in the primal graph.

Lemma 3.1: Consider a triangle Δabc with edge $\{a, b\} \in E_F$. Let o be the owner of Δabc. Then $o = a$ or $o = b$.

Proof: By contradiction: assume $o \neq a$ and $o \neq b$. Then consider the expanding state for Δoab. Since o is the owner in the expanded state, o must have been the navigation robot in the expanding state. Therefore $\{a, b\}$ was the frontier edge in the expanding state and $\{a, b\}$ is now the internal edge in the expanded state, a contradiction. ∎

Theorem 3.2: The owners of two adjacent triangles must also be connected.

Proof: Let Δabc and Δabd be the two adjacent triangles, and $\{a, b\}$ be the edge they share. These two triangles can be formed in the following two ways (in the expanding state): 1) Robot a was the navigation robot. Then a is the owner for both Δabc and Δabd. a is connected to itself. 2) Robot d was the navigation robot. This makes Δabc an existing triangle and $\{a, b\}$ a frontier edge in the expanding state. d is also the owner robot for Δabd in the expanded state. Either a or b is the owner of Δabc by Lemma 3.1, so d, the owner of Δabd, must be connected to the owner of Δabc through either edge $\{a, d\}$ or edge $\{b, d\}$. By symmetry, b is equivalent to a and c is equivalent to d. ∎

C. Dual Graph Navigation

We use the dual graph as a navigation guide for robots in our triangulation. If the destination triangle is known, such as a frontier triangle, then a broadcast message can be used to build a BFS tree suitable for navigation [14]. Our previous work shows there is no lower bound on the competitive factor of the stretch of a path in the online MATP problem [1], but this requires narrow corridors of infinitesimal width. In the following, we show that more realistic assumptions do allow constant-factor performance.

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 V. 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 3.1: 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:

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

This definition is used to prove properties of triangulations.

D. Covered Area

Theorem 3.3: 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 $\sqrt{3}\rho^2/2\sin(\alpha)$ of the optimum.

Proof: Each edge has length between $r_{\min}$ and $r_{\max}$, and any angle is bounded from below by α. Then the area of a triangle is at most $\sqrt{3}r_{\max}^2/2$. On the other hand, it follows from elementary trigonometry that an obtuse triangle has area at least $\sin\alpha r_{\min}^2/2$, while 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 claim follows. ∎

Note that in a practical setting, ρ will be much smaller than the theoretically possible worst case. See Fig. 11b for a real-world evaluation.

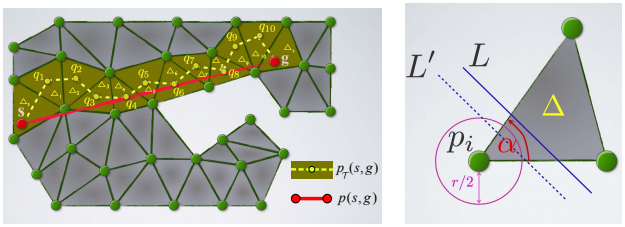
E. Path Stretch

Now we establish that the dual graph of our triangulations can be exploited for provably good routing. We make use of the following terminology.

Definition 3.2: Consider a triangulation $\mathcal{T}$ of a planar region $\mathcal{R}$, with vertex set V. 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 3.4: Consider a (ρ, α)-*fat* triangle Δ with minimum edge length at least r_{min}; let Δ be intersected by a



(a) (b)

Fig. 5: (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 the triangle not too close to one of the endpoints, the length of the intersection is long. If the line passes 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.

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 Fig. 5b. 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.

Theorem 3.5: Consider a (ρ,α)-*fat* triangulation $\mathcal{T}$ of a planar region $\mathcal{R}$, with vertex set V, and maximum and minimum edge lengths r_{max} and r_{min}. 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 $d_{p_{\mathcal{T}}}(s,g) \le c\cdot d_p(s,g)+2$, for $c=\lfloor\frac{2\pi}{\alpha}\rfloor\frac{\rho}{\sin(\alpha/2)}$, and $d_{p_{\mathcal{T}}}(s,g)\le c'\cdot d_p(s,g)$, 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, $\ell'\ge\ell\ge 1$, 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 $d_p(s,g)\ge r_{min}$.

We first show that $d_p(s,g)\ge \ell'\lfloor\frac{2\pi}{\alpha}\rfloor 2\sin(\alpha/2)r_{\min}$. For this purpose, 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 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, $d_{p_{\mathcal{T}}}(s,g)\le 2(\ell+2)r_{max}$. Comparing the lower bound on $d_p(s,g)$ and the upper bound on $d_{p_{\mathcal{T}}}(s,g)$ yields the claim

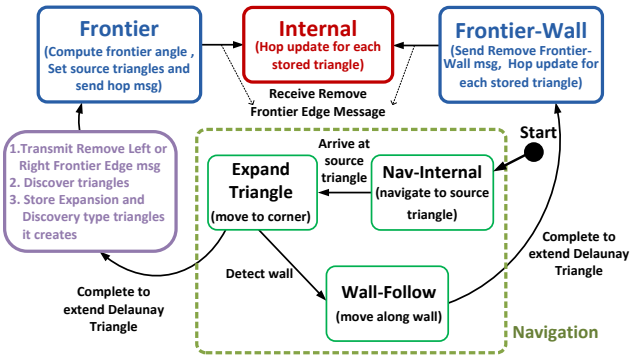


Fig. 6: Finite-state machine for MAT algorithm.

$d_{p_{\mathcal{T}}}(s,g)\le c\cdot d_p(s,g)+2$ with c as stated. The additive term of 2 results from the s and g possibly being close to the boundaries of Δ_s and Δ_g, respectively; it can be removed by noting that $\ell'\ge\ell\ge 1$ implies $(\ell+2)\le 3\ell'$, 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 where the greedy paths do not visit worst-case points in the visited triangles is considerably better, as we show in Section V.

IV. Implementation

A high-level finite-state machine of triangulation construction is shown in Fig. 6. Two robots are initialized in the *Frontier-Wall* state and placed at the base-edge. All other robots begin behind the base edge in the *Navigation* state. Table I lists helper functions for all algorithms below.

A. Navigation 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 Fig. 7a, that returns the current triangle, T_c, that contains robot u, and its owner, o. If T_c is a non-frontier triangle, then u moves to an adjacent triangle that is closer to (fewer hops from) the frontier (lines 10 to 11). Theorem 3.2 ensures that the owner of T_c is connected to owners of adjacent triangles, so u learns the hops of all adjacent triangles with a 2-hop message similar to the geometry message from Fig. 2b. If T_c is a frontier triangle (line 3) or null (only true if u has just crossed the base edge, line 6), then 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).

In the Expand-Triangle state, 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 Fig. 7b until u is in region 3.

We lack the space for a complete description of the controller, so we sketch its operation here. When robot u enters region 3, if $\theta_L>\theta_R$, u first moves toward $B_u(u.L)+\pi$ until $\theta_R\ge\frac{\pi}{3}$. It then changes its heading toward $B_u(u.R)+\pi$,

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 ← GETMINHOPADJTRI(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
```

and moves until it reaches the goal region (region 4). The opposite control happens when $\theta_L < \theta_R$.

Robot u stores the triangle on its list (line 5) and runs the DISCOVERTRIANGLE procedure to discover all adjacent triangles as described in section III-A (line 6). The *Frontier angle*, θ_F, provides a simple way to evaluate the quality of candidate triangles; we define a triangle to be high-quality if $\theta_F < k$, with k manually tuned to reduce errors. Lines 6-11 in algorithm 2 are also related with updating frontier edges, and details will be shown in section IV-D.

If u detects a wall while expanding a triangle (line 12), it changes its state to *Wall-Follow* (line 13). This controller moves u along the wall until it forms an isosceles triangle. Then u stores the triangle, broadcasts disconnect message to $u.L$ or $u.R$, and changes its state to *Frontier-Wall*.

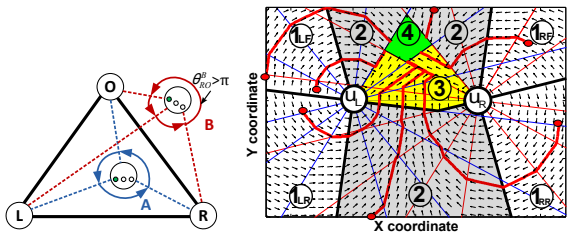
This EXPANDTRIANGLE algorithm uses messages to disconnect and connect triangles from the frontier path. A configuration with triangles that are not (ρ, α)-*fat* could require many messages, but this is unlikely. Our implementation allows for 6 messages, but we only used a max of 2.

B. Frontier and Frontier-Wall State

When robot u enters the *Frontier* or *Frontier-Wall* state, it becomes stationary and runs algorithm 3. In lines 3-7, u labels all of its triangles that include a frontier edge as frontier triangles. These triangles become sources for frontier messages that recruit navigating robots (line 8-9). The frontier robots compute and broadcast the *Frontier angle*, θ_F, between adjacent frontier neighbors, in the direction of the frontier normal, shown in Figs. 3 and 4b. This is done in lines 11-13. The rest of algorithm 3 is in Section IV-D. The algorithm transmits two messages, $O(1)$ complexity.

TABLE I: Table of Helper Functions

GETCURRENTTRIANGLE()	Runs occupancy test and returns current triangle, T_c.
GETMINHOPADJTRI(T_c)	Get T_c's min-hop adjacent triangle.
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()	Broadcast new frontier msg to nbrs.
RECVFMSG()	Receive new frontier nbrs.
UPDATEFNBR()	Change frontier nbrs of u
BCASTDISCONNECTMSG()	Broadcast disconnect msg to nbrs .
RECVDISCONNECTMSG()	Return u_{sender} if u_{sender} disconnects u.
UPDATETRIANGLEHOP(T_i,$N(o)$))	For each triangle u owns, sets its hop to 1 + minimum among all adjacent triangles' hops.
BCASTTRIANGLEHOP($N(o)$)	Broadcast hops of all triangles u owns.



(a) Triangle Occupancy **(b)** Triangle Expansion Controller

Fig. 7: (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 or u_R, 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.

C. Internal State

Robot u is likely to eventually become an *Internal* robot. Every robot relays broadcast messages, updating the hops of each triangle they own by finding the minimum hop at adjacent triangles, and adding one. This propagates the broadcast message, updates the hops, and ensures that any new robot crossing the base edge will move to the frontier triangle that is nearest in the dual graph, providing a breadth-first construction. This uses one message, $O(1)$ complexity.

D. Maintaining the frontier path graph

To ensure 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 u expands a triangle with $\{L, R\} \in E_F$, it updates its internal frontier edges in line 7 of Alg. 2. It then sends a disconnect message to $\{L, R\}$ in line 8, telling them to disconnect from each

Algorithm 3 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
```

other and connect their frontier to u. This message is received in line 17 of Alg. 3. The process for connecting and disconnecting discovered triangles is similar. Robot u updates and broadcasts its frontier neighbors in lines 7-8 of Alg. 2. This message is received on line 17 of Alg. 3, disconnects $\{L_i, L_{i-1}\}$ from each other, and connects L_i to u. This continues for all of $L_i \in N(u)$, eventually leaving 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 u enters the *Frontier-Wall* state and its left (or right) is also in *Frontier-Wall*, then the frontier end-point is disconnected, u disconnects from v and v disconnects from u, making u the new endpoint of P_F.

V. EXPERIMENTAL RESULTS

We have performed several hardware experiments, using the r-one robots shown in [15]. The capabilities of this platform supports the assumptions in our problem statement; each robot can measure the bearings to its nearby robots, despite of a limited resolution of only $\frac{\pi}{8}$, and exchange messages including those bearings and necessary information to run an implemented algorithm in Section IV using inter-robot communication. Each robot also has 8 bump sensors that provide wall detection. To evaluate a resulting triangulation quality or trace the trajectory of a navigation robot, we use AprilTag [16]. 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 Fig. 2b.

A. Maximum Area Trangulation using MATalgorithm

Fig. 8 shows snapshots of triangulation. Over 8 trials using 9-16 robots, the average triangulated area is $1.5\pm0.29m^2$. It takes 7.8±2.1 robots to cover a unit area ($1m^2$). The resulting triangulations are (ρ=3.6, α=0.36 rad)-*fat*. Fig. 9a shows that our triangulations cover about 91% of the region behind the frontier edges. The uncovered region is because

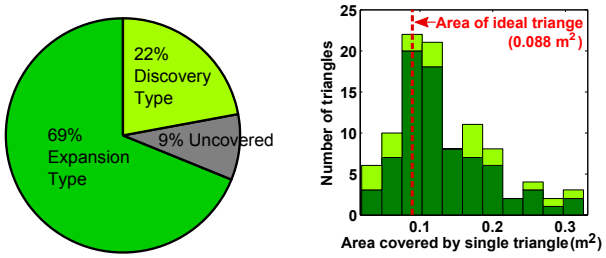
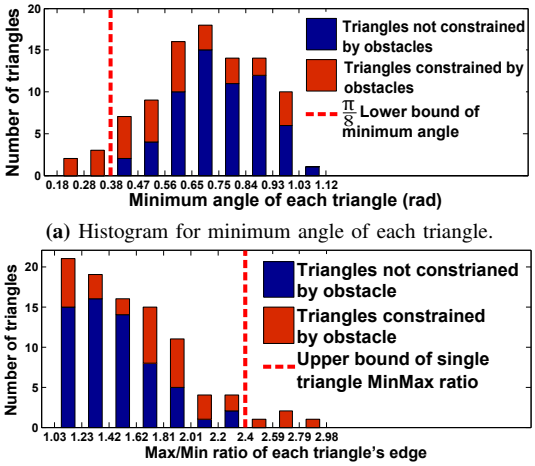


Fig. 9: (a) Pie chart for covered area by all triangles. **(b)** Histogram of covered area by each triangle.



(a) Histogram for minimum angle of each triangle.

(b) Histogram for MaxMin ratio of each triangle.

Fig. 10: (a) Distribution of minimum angle of each triangle, not for a global ρ. All triangles not constrained by a wall satisfy the lower bound, $\frac{\pi}{8}$. **(b)** Distribution of MaxMin ratio of each triangle. All triangles whose minimum angle is larger than the $\frac{\pi}{8}$ (blue colored) also satisfy corresponding upperbound MaxMin ratio.

the top-left and bottom-left corner in Fig. 8 are wall edges (incident on two wall robots), and are not expanded by navigating robots. Fig. 9b shows the distribution of area covered by individual triangles. The initial length of the base edge predicts the area of an ideal equilateral triangle should be $0.088m^2$, our triangles have a mean area of $0.13m^2$, with a std. dev. of $0.065m^2$. This discrepancy 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.

Figs. 10a and 10b show our measurements of individual triangle quality: the distribution of minimum angle and maximum/minimum edge length ratio (MaxMin ratio) for each triangle. The individual data shows 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 1. 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. We leave this for future work.

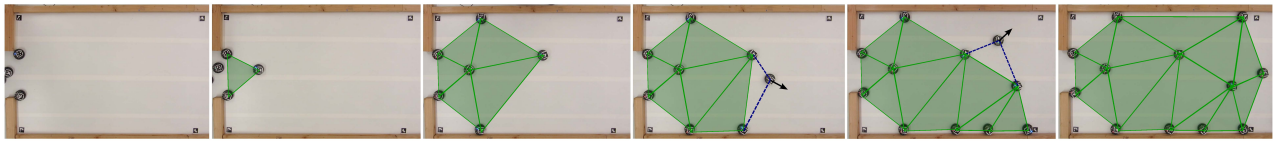
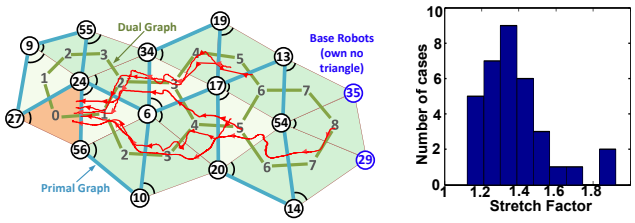


Fig. 8: Screenshots while constructing a triangulation with 12 robots.



(a) (b)

Fig. 11: (a) Sample navigation results with r-one robots. **(b)** Stretch factor histogram with 34 trials of navigation experiment.

B. Dual Graph Navigation

We start each navigation experiment with a constructed triangulation. The triangulation is (ρ=1.36, α=0.88rad)-*fat*. For each trial, we randomly select one triangle as a goal, and the robots build a tree on the dual graph. Fig. 11a shows five trials of the 34 we conducted. The numbers inside the triangles indicate the hops in the dual graph from the goal triangle. The trial from the 8-hop triangle is also shown in Fig. 1. The thick blue lines show connectivity between owners of adjacent triangles. Note that this graph is not complete, but it is a spanning graph of all triangle owners in G, which is implied by Theorem. 3.2.

Fig. 11b shows the distribution of the stretch factor over all trials of navigation tests with various start-goal pairs. The mean stretch factor is 1.38$\pm$0.19. This is much less than the theoretical bound implied by Theorem 3.5, which is based on worst-case assumptions. Our occupancy algorithm correctly returns the triangle including u 91% of the time. Robot u navigates to the correct adjacent triangle 99% of the time. Incorrect navigation is caused by occupancy errors resulting from bearing sensing errors (low-resolution sensors) but not from topological or algorithmic errors.

VI. Conclusion

We presented a distributed algorithm to: triangulate a workspace, produce a physical data structure, and use this structure for communications and robot navigation. There are many exciting new challenges that lie ahead. The next step is to extend this approach with a self-stabilizing algorithm that can construct and repair the triangulation and dual graph dynamically, starting from an arbitrary distribution of robots. While existing controllers can already form triangulated graphs [17], what is needed is construction and maintenance of the physical data structures, i.e., the primal and dual graphs. Future work can extend these ideas to very large populations and dynamic environments. Another objective is to improve the routing algorithm to replace the simple dual graph paths by more sophisticated geodesic trajectories. We are currently working on multi-robot patrolling using the physical data structure to store visitation frequencies and implement a geodesic Lloyd's controller to provide periodic coverage of the triangulation with multiple patrolling robots.

References

[1] S. P. Fekete, T. Kamphans, A. Kröller, J. Mitchell, and C. Schmidt, "Exploring and triangulating a region by a swarm of robots," in *Proc. APPROX 2011*, ser. LNCS, vol. 6845. Springer, 2011, pp. 206–217.

[2] A. Becker, S. P. Fekete, A. Kröller, S. K. Lee, J. McLurkin, and C. Schmidt, "Triangulating unknown environments using robot swarms," in *Proc. 29th Annu. ACM Sympos. Comput. Geom.*, 2013, pp. 345–346, video available at http://imaginary.org/film/triangulating-unknown-environments-using-robot-swarms.

[3] S. P. Fekete and A. Kröller, "Geometry-based reasoning for a large sensor network," in *Proc. 22nd Annu. ACM Sympos. Comput. Geom.*, 2006, pp. 475–476.

[4] M. Bern and D. Eppstein, "Mesh generation and optimal triangulation," in *Computing in Euclidean Geometry*, ser. Lecture Notes Series on Computing, D.-Z. Du and F. K. Hwang, Eds. Singapore: World Scientific, 1992, vol. 1, pp. 23–90.

[5] A. Efrat, S. P. Fekete, P. R. Gaddehosur, J. S. Mitchell, V. Polishchuk, and J. Suomela, "Improved approximation algorithms for relay placement," in *Proc. 16th Annu. Europ. Sympos. Algor.* Springer, 2008, pp. 356–367.

[6] J. Bredin, E. Demaine, M. Hajiaghayi, and D. Rus, "Deploying sensor networks with guaranteed fault tolerance," *Networking, IEEE/ACM Transactions on*, vol. 18, no. 1, pp. 216 –228, feb. 2010.

[7] T.-R. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and J. S. B. Mitchell, "Algorithms for rapidly dispersing robot swarms in unknown environments," in *Algorithmic Foundations of Robotics V*, ser. STAR, J. B. et al., Ed., vol. 7. Springer, 2004, pp. 77–93.

[8] J. McLurkin and J. Smith, "Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots," in *Proc. 7th Internat. Sympos. Distr. Auton. Robot. Syst.*, 2004.

[9] K. Konolige, D. Fox, C. Ortiz, A. Agno, M. Eriksen, B. Limketkai, J. Ko, B. Morisset, D. Schulz, B. Stewart *et al.*, "Centibots: Very large scale distributed robotic teams," in *Experimental Robotics IX*. Springer, 2006, pp. 131–140.

[10] J. W. Durham, A. Franchi, and F. Bullo, "Distributed pursuit-evasion with limited-visibility sensors via frontier-based exploration," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 3562–3568.

[11] A. Ghoshal and D. A. Shell, "Being there, being the rrt: Space-filling and searching in place with minimalist robots." in *AAAI Spring Symposium: Multirobot Systems and Physical Data Structures*, 2011.

[12] W. M. Spears, D. F. Spears, R. Heil, W. Kerr, and S. Hettiarachchi, "An overview of physicomimetics," *Lecture Notes in Computer Science-State of the Art Series*, vol. 3342, 2005.

[13] J. McLurkin, "Analysis and implementation of distributed algorithms for Multi-Robot systems," Ph.D. thesis, Massachusetts Institute of Technology, 2008.

[14] Q. Li and D. Rus, "Navigation protocols in sensor networks," *ACM Trans. Sen. Netw.*, vol. 1, no. 1, pp. 3–35, 2005.

[15] J. McLurkin, A. J. Lynch, S. Rixner, T. W. Barr, A. Chou, K. Foster, and S. Bilstein, "A low-cost multi-robot system for research, teaching, and outreach," in *Distributed Autonomous Robotic Systems*, vol. 83. Springer Tracts in Advanced Robotics, 2013, pp. 597–609.

[16] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.

[17] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil, "Distributed, Physics-Based control of swarms of vehicles," *Autonomous Robots*, vol. 17, no. 2, pp. 137–162, 2004.