Continuous Sculpting: Persistent Swarm Shape Formation Adaptable to Local Environmental Changes

Andrew G. Curtis[1], Mark Yim[2], Michael Rubenstein[1]

Abstract—**Despite their growing popularity, swarms of robots remain limited by the operating time of each individual. We present algorithms which allow a human to sculpt a swarm of robots into a shape that persists in space perpetually, independent of onboard energy constraints such as batteries. Robots generate a path through a shape such that robots cycle in and out of the shape. Robots inside the shape react to human initiated changes and adapt the path through the shape accordingly. Robots outside the shape recharge and return to the shape so that the shape can persist indefinitely. The presented algorithms communicate shape changes throughout the swarm using message passing and robot motion. These algorithms enable the swarm to persist through any arbitrary changes to the shape. We describe these algorithms in detail and present their performance in simulation and on a swarm of mobile robots. The result is a swarm behavior more suitable for extended duration, dynamic shape-based tasks in applications such as entertainment, agriculture, and emergency response.**

Index Terms—**Swarms; Path Planning for Multiple Mobile Robots or Agents; Distributed Robot Systems; Shape Formation**

I. Introduction

As their popularity continues to grow, commercial drones are employed in swarms more and more frequently. Swarms of flying robots are commonly used in shape-based applications, such as entertainment drone shows, where robots navigate through a set of predefined waypoints to form different shapes. Other shape-based flying swarm applications include search and rescue, emergency response communications networks, and crop monitoring, where a flying swarm maintains a 2D formation over an area of land to perform its task.

The challenge with most shape-based flying swarm applications is the limited flight time, or endurance, of the robots. Typically, robots must land to recharge, and the task is paused until the robots can fly again (e.g., crops go unmonitored while the robots charge). In addition, most swarms only form static predefined shapes, limiting their potential applications.

Manuscript received: January 5, 2024. Resubmitted: May 13, 2024.
This paper was recommended for publication by Editor TBD.
This work was supported by the NDSEG Fellowship and The National Science Foundation, NRI2.0 grants 2024692 and 2024615.

[1]Andrew G. Curtis and Michael Rubenstein are with the Center for Robotics and Biosystems, McCormick School of Engineering, Northwestern University, Evanston, IL USA `agc@u.northwestern.edu`, `rubenstein@northwestern.edu`.
[2]Mark Yim is with the General Robotics, Automation, Sensing and Perception (GRASP) Lab, University of Pennsylvania, Philadelphia, PA USA `yim@seas.upenn.edu`.

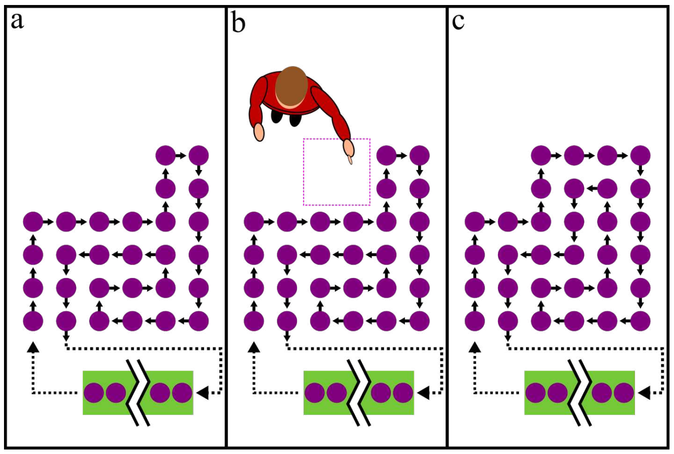

Fig. 1: Cartoon overhead view of adaptive and persistent shape formation. (a) A swarm of mobile robots (purple circles) leave a charging station (yellow box) to enter and form a shape. The robots move through the shape in the directions indicated by the black arrows until they exit the shape and return to the charging station. (b) A human points to where they would like to add to the shape (purple box). (c) The swarm adjusts to form a path through the new shape while continuing to cycle to and from the charging station.

To overcome these challenges, we present a novel approach to persistent and adaptable shape formation. We shift the paradigm of shape formation from shapes formed by static robots to shapes formed by a sequence of robots moving along a path. The path allows robots to cycle to and from a charging station, so robot endurance is no longer a constraint on shape duration. The path is also adaptable to external environmental stimuli (e.g., human interactions) so that the shape is free to change over the duration of the task. For example, farmers could form a swarm of drones into a shape over their fields to persistently monitor crops (regardless of individual robot charge). They could then directly interact with the swarm to change the shape of the swarm and thus the fields of crops being monitored. Similarly, a performer could interact with a drone show, changing the shape of the swarm to entertain an audience beyond the flight time of individual flyers.

We achieve shape persistence by allowing swarm robots to cycle between a shape and a charging station indefinitely via a path that both approximates the shape and facilitates the movement of robots through the shape and back to a charging station (Fig. 1). With each change to the shape, the swarm adapts to another path and the process continues. Our primary contributions, the algorithms we present for both shape persistence and shape adaptability, are decentralized and scalable to large swarms. They also open the door to a new application of human-swarm interaction - continuous sculpting - where a human can use intuitive gestures to actively morph a swarm into a persistent shape in a manner similar to the way an artist might morph clay into a sculpture. One potential issue with continuous sculpting is making sure potential collisions with humans are safe, but this can be solved by making robots small and light [1].

The remainder of this paper is structured as follows. After discussing related work (Section II), we describe the basic robot capabilities required for a swarm to execute the algorithms (Section III). Then, we introduce the algorithm responsible for shape persistence, called the **default behavior**. The default behavior is introduced and demonstrated in Section IV, and the theory supporting the algorithm is described in Section V. The later portions of the paper are dedicated to the algorithms responsible for shape adaptability which are only executed to resolve a shape change. The adaptability algorithms are introduced in Section VI. In Section VII, we provide demonstrations of both adaptability and persistence in the presence of humans and for a large swarm. Finally, in Section VIII we describe the theory supporting the adaptability algorithms, and in Section IX we summarize our work and describe our future plans. Terms defined in the text are indicated in **bold** and their definitions are included in Appendix A.

II. Related Work

This work lies in a largely unstudied area of swarm robotics with few direct state-of-the-art comparisons. However, related works exist in the fields of shape formation, area coverage, robot recharging, Hamiltonian path planning, and human-swarm interaction.

A. *Shape Formation*

One approach to persist in a shape is to have individual swarm robots cycle into and out of the shape to recharge. Here, the energy limitations of individual robots no longer constrain shape duration but instead constrain the size of the path that an individual robot can travel and thus the size of the final shape. One problem occurs if a robot is ever “stuck” along its path in the middle of the shape formation, unable to return to a charging station because it is surrounded by other robots forming the shape. Stuck robots are a potential problem for most static shape formation algorithms that begin with robots in some arbitrary starting position and end with robots statically holding a position in the defined shape [2]–[7].

However, there are some non-static shape formation algorithms that use dynamic shape scaling [8], time varying shape formations [9], and robot position swapping [10] so that robots are not always statically holding a position in the defined shape. These works are likely the closest existing shape formation works to our algorithms as they enable swarm shape formation in such a way that could potentially be altered to include robot recharging. However, they either lack guarantees that a particular robot will never become stuck or they operate on predefined shapes and therefore lack adaptability to real-time changes. Some programmable matter shape formation algorithms have shown similar potential as they have displayed the ability to distributively change from one shape to another [11], [12], but these too lack guarantees that robots will never become stuck.

B. *Area Coverage*

Traditional area coverage algorithms are similar to adaptive and persistent shape formation because they can cycle robots through a defined shape or area, though typically at a lower density of robots than most shape formation algorithms. Dozens of area coverage algorithms already exist for aerial vehicles [13]. Many of these algorithms use a centralized controller for traditional back-and-forth path planning [14]–[17] or spiral-like path planning [18]. Still others implement decentralized approaches (which are more suitable for swarms) using built-in randomness [19], cost functions [20], and virtual pheromones [21]. However, the decentralized approaches typically result in nonuniform area coverage which, if used for shape formation, would result in distorted shapes with no guarantees for avoiding stuck robots.

C. *Swarm Robot Recharging*

Others have pushed swarm performance beyond the battery lifetime of individual robots by either: 1) continuous charging where power is brought to the robots as they operate [22] or 2) robot swapping where individual robots take turns recharging at designated charging stations [23]–[27]. Our work falls in group 2. Although most works in this group are concerned with swarm tasks like exploration [23], [24] where the swarm need not maintain a particular formation, there are a few associated with shape formation. However, these swarms operate on either fixed formations [25] that are not adaptable to environmental changes or they use centralized planners to optimize robot movements [27]. To the authors’ best knowledge, this work is the first of its kind to incorporate robot recharging and shape adaptability for robot swarms in a decentralized way.

D. *Hamiltonian Path Planning*

In this work, we ensure that a robot can cycle through a 2D shape and back to a charging station by using planar (i.e., non-intersecting) Hamiltonian cycles. These guarantee the robot’s path covers the entire shape uniformly. If the cycle is further constrained to have adjacent start and end positions on the periphery of the shape, then the robot is guaranteed to be able to cycle into and out of the shape without becoming stuck or colliding with its peers. Unfortunately, finding a Hamiltonian cycle through a shape represented as a grid graph of points in space is a NP-complete problem [28]–[30], making it difficult to scale to large shapes.

Despite the challenge, others have found Hamiltonian cycles by either restricting the problem to grid graphs that are solid (i.e., no holes) [29] or for specialized purposes such as programmable chain assembly [31], obstacle avoidance [32], and nonuniform aerial coverage [33]. However, in all of these approaches, Hamiltonian cycles are found via a centralized planner using global information about the entire grid graph or a single robot moving through (and sensing) the entire environment. In a swarm setting, it is important that Hamiltonian cycles are generated in an online, decentralized manner with each individual robot planning its next step using only local information. This helps the swarm stay adaptable and permits the use of simple robots.

E. Human-Swarm Interaction

Kolling et al. [34] define humans directly interacting with a swarm in a shared environment as proximal interaction. Examples of proximal interaction include works where swarms are shown to detect and identify specific human gestures [35], [36]. In our work, we use human gestures in a shared environment to initiate shape changes that modify (i.e. sculpt) the swarm into different shapes. We focus on the swarm's response to the gesture, not necessarily the detection and identification of the gesture itself.

III. Robot Capabilities and Assumptions

The algorithms presented in this paper are agnostic to specific swarm implementations, so the algorithms can be employed on robots with various hardware and software designs. However, there are some basic capabilities we assume each robot has in order to execute the algorithms.

First, we assume that each robot can store a representation of a grid graph overlay on the environment ($G = (n, e)$). We also assume that each robot can identify if it is residing at a particular node (n) or translating along a particular edge (e) of G. Further, we assume that each robot can identify a subset of grid graph nodes that represent the shape (S) that the swarm is forming in space ($S \subset G$). For the simulations and physical demonstrations in this paper, we elected (by design choice) to have robots maintain a list of in-shape grid nodes. If the shape is ever changed, robots communicate the change using neighbor-to-neighbor communication to propagate the change through the swarm so each robot can update its list.

For this to occur, we also assume that robots are capable of robot-to-robot communication and are equipped with internal clocks. We also assume that each robot can communicate with its peers within a range of $\sqrt{2} * l$ where l is the length of an edge in G. This allows robots to communicate with up to eight immediate neighbors (i.e., N, NE, E, SE, S, SW, W, and NW). During operation of the algorithms, robots use neighbor-to-neighbor communication to inform one another of shape changes, share position information, and synchronize their clocks so that robots can move together in lockstep.

Additionally, since the algorithms in this paper use planar Hamiltonian cycles (and since finding a Hamiltonian cycle in a grid graph is an NP-complete problem), it is not realistic to assume that a given robot can find a Hamiltonian cycle as it navigates through a very large shape. Therefore, we enforce a set of rules to standardize shape structures so that the swarm can more easily find Hamiltonian cycles, including for very large shapes and shapes with holes.

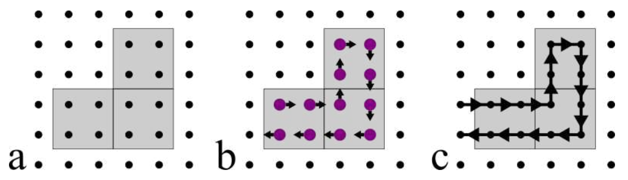

Fig. 2: (a) An example of a valid shape of 3 boxes. Points indicate nodes of G. (b) Robots (purple) approximating the shape. Arrows indicate robot heading. (c) The path along which robots travel. Arrows indicate edge direction.

The rules define a unit shape, called a **box**, that is comprised of four grid nodes in a 2x2 square (Fig. 2). This is similar to the method of finding Hamiltonian paths by breaking a box down into four smaller boxes, as discussed for space filling curves in [31]. We assume that each robot associates a given 2x2 square of grid nodes with the same box as every other robot. This ensures that a given shape representation is consistent for all robots in the swarm. Further, we assume each robot can determine the box that it is in and its relative position with respect to the center of that box. This allows robots to detect if they are moving clockwise or counter-clockwise around the center of a box.

Allowable shapes are continuous so that each box meets the full side of another box: partial side connections and vertex-to-vertex only connections are not allowed. Partial boxes are also not permitted. Shapes that meet these conditions are called **valid shapes**, and they are important because they reduce the difficulty of finding a planar Hamiltonian cycle from NP-complete to something that is solvable in linear time (run time, $O(\mathcal{B})$, scales linearly with $\mathcal{B}$, where $\mathcal{B}$ is the number of boxes in the shape). In fact, we will show that a planar Hamiltonian cycle can always be found for any valid shape in Section V-B. Finally, although we only consider valid shapes created in a square lattice graph, it is plausible that the algorithms in this paper could extend to other planar graphs, such as hexagonal lattices, but this is reserved for future work.

IV. Shape Persistence

For a swarm to form a persistent shape beyond the energy limitations of any individual robot, the swarm must cycle robots back and forth between the shape and a charging station. This cycle can be separated into four subroutines: 1) forming the shape, 2) traveling from the exit point of the shape to a charging station, 3) charging at the charging station, and 4) traveling from the charging station to the entry point of the shape. The primary contributions of this work are in the first subroutine: forming the shape. To do this, each robot will travel along the same planar Hamiltonian cycle to both approximate the shape and move through the shape without colliding with other robots in the swarm.

To be consistent with prior works, we have adapted the formal definition of Hamiltonian circuits from the 1976 paper by Garey et. al. [28] which first identified the planar Hamiltonian circuit problem as NP complete (Definition 1). We will use this definition throughout the remainder of this paper.

Definition 1 (Planar Hamiltonian Cycle)**.** A planar Hamiltonian cycle in a graph is a path which passes through every vertex exactly once and returns to its starting point without intersecting itself.

Further, in our work, planar Hamiltonian cycles are broken at the adjacent entry and exit points (there are no path edges connecting the two). Thus, the paths that robots create through shapes are technically planar Hamiltonian paths (not cycles). However, for the remainder of this paper, a planar Hamiltonian path with adjacent start and end nodes will be treated identically to a planar Hamiltonian cycle since the former can always be turned into the latter by simply connecting the adjacent start and end nodes. Both will be referred to as Hamiltonian cycles to distinguish from other Hamiltonian paths that do not have adjacent start and end nodes. Finally, we restrict the start and end nodes to the periphery of the shape so that each robot can enter and exit the shape and ultimately cycle to and from a charging station.

A. Subroutine 1: Forming the Shape

In subroutine 1, robots form a planar Hamiltonian cycle to both approximate the shape and cycle robots from the entry node (a grid node on the periphery of the shape where robots enter the shape) to the exit node (a grid node on the periphery of the shape where robots exit the shape to return to the charging station). To do this, each robot enters the shape at the entry node and then visits each grid node in the shape exactly once before finally exiting the shape at the exit node without ever intersecting its own path. Robots keep track of the boxes they have visited V_B and the nodes they have visited V_n while traversing through the shape. Robots follow a set of rules called the **default behavior** when traveling in the shape to determine the order in which grid nodes are visited between the entry and exit nodes using only local information.

The default behavior prioritizes a robot's movement through a shape so that a robot visits new boxes in a manner that mimics a traditional depth-first search. If a robot were performing a depth-first search of the boxes in a shape, the robot would first identify all of the children of its current box. Next, the robot would enter a child box and continue to move parent-to-child as it explored all of the descendant boxes within that subtree. Only after the robot exhausted all descendants in the subtree would the robot move "up" the tree, revisiting boxes in the reverse order (child-to-parent) and repeating the process of visiting descendant boxes before visiting other children. Once all boxes had been visited, the robot would exit the shape.

Similar logic governs the default behavior. As a robot traverses through the shape, it moves from node-to-node such that it visits boxes in a depth-first, clockwise priority manner starting at the entry node and finishing at the exit node. Specifically, when a robot is at a node in the shape, there is a set of four possible edges that the robot could take to move away from that node (i.e., e_N, e_S, e_E, e_W). If the robot is at the exit node, it will take the edge that leads the robot out of the shape to the charging station. For all other nodes (n) in the shape (S), the robot selects which edge to take next by first ignoring any edge that leads the robot to a node that is out of the shape or to a node that the robot has already visited. Then, the following rules are applied to the remaining edges in the set (as depicted in Algorithm 1):

IF an edge leads to a node in a box not previously visited, take that edge. (Rule 1)
ELSE IF an edge leads the robot clockwise within its current box, take that edge. (Rule 2)
ELSE IF an edge leads to a node in the parent box in the tree, take that edge. (Rule 3)

Algorithm 1 Default Behavior

```
Input: $b, S, V_B, V_n$                          ▷ $b$ = current box
Output: $\epsilon'$                              ▷ $\epsilon'$ = edge to next node
 1: if robot at exit node then
 2:     exit shape
 3: else
                 ▷ Create lists of plausible next edges and nodes
 4:     $\mathcal{E} \leftarrow [e_N, e_S, e_E, e_W]$
 5:     $\mathcal{N} \leftarrow [n_N, n_S, n_E, n_W]$
                 ▷ Remove visited and out-of-shape options
 6:     for $\epsilon, n$ in $\mathcal{E}, \mathcal{N}$ do
 7:         if $n \in V_n$ then
 8:             remove $\epsilon, n$ from $\mathcal{E}, \mathcal{N}$
 9:         else if $n \notin S$ then
10:             remove $\epsilon, n$ from $\mathcal{E}, \mathcal{N}$
11:         end if
12:     end for
                 ▷ Employ default behavior rules
13:     if $\exists\ \epsilon, n \in \mathcal{E}, \mathcal{N} \mid$ Box($n$) $\notin V_B$ then      ▷ Rule 1
14:         $\epsilon' \leftarrow \epsilon$                                ▷ Rule 1
15:     else if $\exists\ \epsilon, n \in \mathcal{E}, \mathcal{N} \mid n$ in $b$            ▷ Rule 2
16:                     and $\epsilon$ is clockwise then               ▷ Rule 2
17:         $\epsilon' \leftarrow \epsilon$                                ▷ Rule 2
18:     else if $\exists\ \epsilon, n \in \mathcal{E}, \mathcal{N} \mid n$ in parent of $b$ then ▷ Rule 3
19:         $\epsilon' \leftarrow \epsilon$                                ▷ Rule 3
20:     end if
21: end if
```

While executing the default behavior, robots communicate timing information to synchronize their clocks. This allows robots to move in lockstep with one another, avoiding in-path collisions and maintaining spacing such that one robot is always occupying one grid node in S. Robots also avoid collisions by never crossing another robot's path. This is the result of every robot following the same default behavior rules (and Hamiltonian cycle). Since the default behavior creates a Hamiltonian cycle such that each robot terminates its path at the exit node (which is on the periphery of the shape), and every robot can traverse the shape without collision, we can guarantee that all robots will be able to exit the shape to recharge (i.e., no "stuck" robots), which facilitates the persistence of the shape. The fact that the default behavior results in a planar Hamiltonian cycle is proven in Section V-C.

One key aspect of the default behavior is that robots do not plan their paths through the shape prior to entering it. Instead, each robot makes its own decisions to reactively build a path through the shape from entry to exit one step at a time. That means that our design choice to have robots maintain a list of in-shape grid nodes is not necessary for algorithm operation. If a robot were capable of distinguishing between in-shape and out-of-shape nodes with its onboard sensors, then the list of in-shape grid nodes (S) could be replaced with a list that the robot senses and creates as it moves. This more sophisticated sensing scheme would replace the shape memory requirements in the default behavior without changing the rules for robot motion. This would further expand the practical applications of persistent shape formation to very large shapes (with many grid nodes) or for very simple robots (with limited memory capacity). Investigating this is reserved for future work.

B. Subroutines 2, 3, & 4: Robot Recharging

Although the remaining subroutines are not the primary focus of this work, they are necessary for the swarm's success, so very simplistic behaviors are implemented. The two "traveling" subroutines (traveling to the charging station and traveling to the shape) are simplified by a set of predefined, static waypoints from the exit node to the charging station and from the charging station to the entry node, respectively. This ensures that robots can travel between the charging station and the shape without encountering obstacles or collisions.

The traveling subroutines are further simplified such that robots arrive at the shape and exit the shape at the same user defined constant time interval as robots moving in lockstep between nodes within the shape (e.g., $\tau = 10$s). Further, if the robots in the shape are ever holding still to complete some portion of an algorithm, then the last robot to have left the shape will carry a message to the other robots outside of the shape. This message contains a list of nodes added to or removed from the shape so that the robots outside of the shape can update their list of in-shape nodes. Robots in receipt of this message also adjust their shape-arrival times to limit the number of robots queuing at the shape entry node.

Similarly, the entry and exit nodes are simplified to be static and known *a priori*. By definition, the entry and exit nodes are adjacent to one another on the periphery of the shape and within the same box. Having static and predefined entry and exit nodes allows us to demonstrate the shape persistence aspects of the algorithms without unnecessarily complicating swarm behavior. Changing the entry and exit nodes is reserved for future work.

The charging subroutine is also simplified since most implementations would be highly dependent on the type of charging technology used (e.g., wireless charging vs. contact charging). For this work, the charging station is modeled as a set of static positions in space. Upon reaching a position in the charging station, a robot begins charging. When a robot is done charging, it leaves the charging station to travel back to the shape in a first-in-first-out order and such that it will arrive at the shape at the constant time interval (τ).

In addition to these simplifications, we assume that robots move through the entire shape and back to the charging station on one charge and that the swarm has enough robots that each robot has more than enough time to completely recharge before it cycles back into the shape. If for a given implementation a shape is too large that robots could not pass through the shape and back to the charging station on a single charge, a system of multiple charging stations could be implemented, effectively breaking the shape into multiple pieces. Similarly, if robots do not have enough time to completely recharge before cycling back into the shape, one could either add more robots (and more charging stations) or reduce charging time. Investigating these scenarios are reserved for future work.

C. Demonstrations of Persistence

Shape persistence was demonstrated in identical tests for both a simulated swarm and a swarm of physical robots. In both kinds of experiments, a user initialized a shape of four boxes (with a start and end node) in a grid graph with edge lengths of $l = 0.2$m. 16 robots were set up in a temporary queue outside of the shape, and 22 robots were placed in a charging station. Coordinates of the charging station, as well as the waypoints to and from the charging station, were preprogrammed and provided to all robots. These coordinates did not change during the course of the experiment. Communication was artificially limited to a range of 0.3m for each robot.

After initialization, the robots in the temporary queue began to fill the shape via the default behavior. Each robot stepped in synchronization with other robots (due to clock synchronization messages), moving to a new node every $\tau = 12$s. Once the shape was almost completely formed, robots in the charging queue began to leave so that they would arrive at the shape in $\tau = 12$s increments starting as soon as the first robot was about to exit the shape. For every 12s after that, one robot left for the charging station, one robot arrived from the charging station, and each robot within the shape moved to its next node. At that point, the system was in steady state and allowed to persist for some time.

In both the simulation and physical experiments, the charging station was modeled as a set of linear positions along one side of the environment. This was meant to emulate a "charging rail" where robots could pull up to charge themselves. This design decision did create the potential for collisions between robots entering a charging station position and robots exiting a charging station position. Robots communicated their own position to one another to help avoid collisions. In the event of a potential collision, robots heading to the charging station yielded to robots heading toward the shape to ensure the shape-bound robots arrived at the shape on time. Furthermore, robots entered and exited the charging station in a first-in-first-out basis.

For the physical robot demonstration, we used an existing swarm of ground robots called *Coachbot V2.0* [2]. Each *Coachbot* is equipped with a two-wheeled differential drive system and the ability to sense its position (x, y, θ) in the arena. The robots are also equipped with an onboard battery, an onboard Raspberry Pi computer, and onboard Wi-Fi for robot-to-robot communication that can be artificially limited to a user defined distance. The simulator was built to emulate the performance of the *Coachbots* (i.e., differential drive robots with the ability to sense x, y, and θ, robot-to-robot communication, etc.).

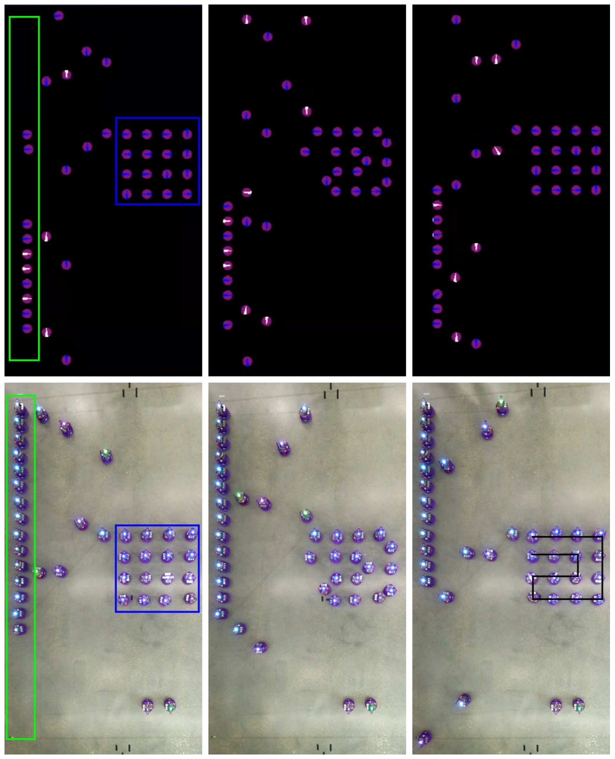

Fig. 3: Images from a simulation (top) and physical robot experiment (bottom). Each set of three images represents a sequence in time of the robots forming the shape (left), stepping to their next node (center), and holding at their next node (right). The charging station is highlighted by the green box, and the desired shape is highlighted by the blue box. Robots in neither box are cycling to or from the shape. The resulting path through the shape for both the simulation and the physical robots is shown in the bottom right image. The LED color of each robot is insignificant for these experiments.

Images from both the simulation and physical robot experiments can be found in Fig. 3. Both the simulated swarm and the physical swarm were left to maintain their respective shape for over 30 minutes (and some simulations were left to run for well over an hour). In that time, each robot cycled through the shape and back to the charging station 4 or 5 times (9 or 10 times for the hour-long simulations). The experiments were stopped by an experimenter, not due to robot or algorithmic failure. This suggests that each swarm is capable of persistent shape formation for extended duration tasks because both the simulated swarm and the physical swarm demonstrated the ability to cycle robots through the shape via the default behavior without failure or collision for multiple cycles through the entire swarm of robots.

The *Coachbots* were used for the physical demonstrations because they are an established swarm platform that was available to the authors. However, as ground robots, they could simply drive to form a shape, power down, and persist in that shape indefinitely without needing to recharge. In the future, we plan to demonstrate swarm persistence with a swarm of flying robots (currently under construction) that do not have the luxury of a floor to help them hold position in space. In the meantime, the *Coachbots* served as an experiment-ready option that enabled us to demonstrate the algorithms on a physical platform.

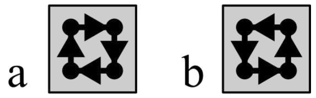

Fig. 4: (a) A clockwise planar Hamiltonian cycle through a unit shape. (b) A counter-clockwise planar Hamiltonian cycle through a unit shape. Arrows indicate edge direction.

V. Shape Persistence Theory

The theory behind the default behavior is rooted in some provable properties of valid shapes and planar Hamiltonian cycles. The remainder of this section formally addresses this theory in three parts: establishing basic concepts, proving planar Hamiltonian cycles can be found for any valid shape, and proving the default behavior always produces a planar Hamiltonian cycle.

A. Basic Concepts

In order to simplify the discussion of shape persistence theory, we define some basic terminology and concepts. While most of these concepts are used in the shape persistence proofs, some of them are also applicable to the shape adaptability concepts discussed in later sections.

The first concept is the **unit shape**: a shape of $b = 1$ box. A robot can take one of two possible cycles through a unit shape: clockwise or counter-clockwise (Fig. 4). By design choice, we elect to use the clockwise cycle as default in this paper. We define a clockwise planar Hamiltonian cycle through a unit shape as the **unit path**. Choosing the counter-clockwise cycle as the unit path would have also been valid. In that case, the algorithms would result in robots moving along paths (and edges) in the opposite directions as the ones shown in this paper. Furthermore, Algorithm 1 rule 2 would have to be changed to prioritize counter-clockwise motion instead of clockwise. The remainder of the concepts and theories presented in this paper would go unchanged.

In discussing the remaining concepts, we will use the following terminology: **parallel** to refer to path edges that are equidistant everywhere and do not intersect; **opposite** to refer to edges that have directions exactly 180° from one another; **spanning** to refer to path edges that begin and terminate in different boxes; and **non-spanning** to refer to path edges that begin and terminate in the same box. Finally, we will define two path edges as a **pair** if the start and end nodes of both path edges are in a 2x2 grid configuration (regardless of if that grid configuration is within a box or across multiple boxes). Thus, the unit path is comprised of two pairs of parallel and opposite non-spanning edges (a pair directed North-South and a pair directed East-West).

With these terms, we can discuss merging two paths into one or separating a single path into multiple and, in the process, establish two lemmas.

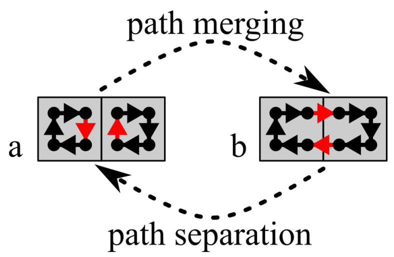

Fig. 5: The fundamental operations of path merging (going from a to b) and path separation (going from b to a). Red indicates impacted edges. Arrows indicate edge direction.

Lemma 1. *If path merging is performed on two separate planar Hamiltonian cycles, then the result will be one combined planar Hamiltonian cycle.*

Lemma 2. *If path separation is performed on a planar Hamiltonian cycle, then the result will be separate planar Hamiltonian cycles on either side of the box connection where the separation occurred.*

The first concept, **path merging**, involves separate paths merging into one path by creating a pair of spanning edges across a box connection. During path merging, a pair of parallel and opposite non-spanning edges (one in each box on either side of the box connection) are replaced with a pair of parallel and opposite spanning edges (Fig. 5a to b). The resulting path is a planar Hamiltonian cycle through all of the nodes in the combined path by Definition 1. Further, the only edges affected by this change are the pairs of edges that go from non-spanning to spanning; the rest of the edges and nodes are unchanged. Therefore, it does not matter if the pre-merge paths were simple planar Hamiltonian cycles (such as the unit paths shown in Fig. 5a), or if the pre-merged paths were larger, more complicated planar Hamiltonian cycles. Because the changed portions of the path are within Definition 1, and since all other portions are unaffected, the result is still a planar Hamiltonian cycle. Thus, we can conclude Lemma 1.

The opposite concept, called **path separation**, involves one planar Hamiltonian cycle separating into multiple paths. For this to occur, a pair of parallel and opposite spanning edges are replaced with a pair of parallel and opposite non-spanning edges (Fig. 5b to a). The result is two planar Hamiltonian cycles on either side of the box connection after separation. Similar to path merging, the only edges affected are the pair of edges that go from spanning to non-spanning; the remainder of the shape nodes and path edges from the pre-separated planar Hamiltonian cycle are unchanged. Therefore, it does not matter if the post-separated paths are simple unit paths, as shown in Fig. 5a, or if the post-separated paths are larger, more complicated planar Hamiltonian cycles. In either case, path separation will always result in planar Hamiltonian cycles on either side of the box connection after the separation. Thus, we can conclude Lemma 2.

B. Planar Hamiltonian Cycles in Valid Shapes

Earlier, we claimed that a planar Hamiltonian cycle can always be found for any valid shape, and we suggested that

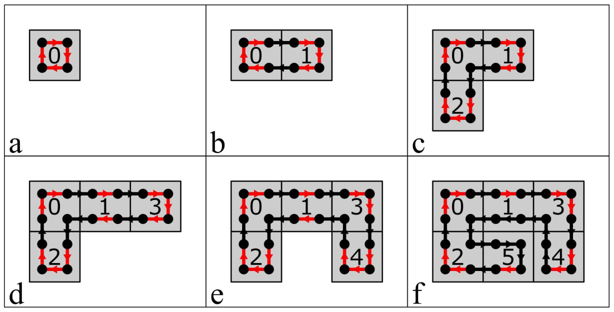

Fig. 6: An arbitrary sequence of shape construction (a-f). Each additional box creates a new valid shape and valid path. Periphery edges are red. Arrows indicate edge direction.

the process of finding a cycle would scale linearly with the number of boxes in the shape ($O(\mathcal{B})$ scales linearly with $\mathcal{B}$). To prove these claims, we establish four things: the assembly of valid shapes by sequential box addition, path merging during shape assembly, **periphery nodes**, and **periphery edges**.

First, consider the assembly of valid shapes. Since valid shapes are constructed from boxes, any valid shape can be assembled by adding boxes together in an arbitrary sequence until the desired final shape is achieved. For example, the shape in Fig. 6f could be constructed by assembling boxes in the order 0,4,3,2,1,5 or 0,1,2,3,4,5, etc. We can further constrain assembly so boxes are only added together such that a valid shape is maintained after each addition (i.e., each additional box is added to be adjacent to an existing box). To continue the example under such restrictions, 0,4,3,2,1,5 would no longer be a valid sequence for assembling the shape in Fig. 6f, but 0,1,2,3,4,5 would be valid because it maintains a valid shape after each additional box.

Next, consider the concept of path merging during shape assembly under the valid shape constraint. Every time a box is added to a valid shape, its unit path could be merged with the existing path via path merging. By Lemma 1, this would result in a new planar Hamiltonian cycle through the shape. For example, when box 2 is assembled with boxes 0 and 1 in Fig. 6c, the unit path in box 2 is merged with the existing path in boxes 0 and 1. Specifically, the bottom edge of the path in box 0 and the top edge of the unit path in box 2 are the pair of parallel and opposite non-spanning edges that are turned into spanning edges to merge the paths. The result is a new planar Hamiltonian cycle through boxes 0, 1, and 2.

Finally, we define a **periphery node** as an in-shape node on the periphery of the shape and a **periphery edge** as a non-spanning edge between two periphery nodes. Periphery edges represent edges that could be merged with unit paths when new boxes are added to the shape. If every periphery node is connected by at least one periphery edge, then a new box could be added anywhere along the periphery of the shape and its unit path could always merge with the existing path. For example, every periphery node in the shape in Fig. 6c is connected by at least one periphery edge, so a new box could be added to any side on the periphery of the three-box shape.

Theorem 1. *A planar Hamiltonian cycle exists for any valid* *shape.*

Proof 1. This is a proof by induction. Consider the base case of the unit box ($\mathcal{B} = 1$). This shape has three important properties. Trivially, it is a valid shape (property 1) and its path is a planar Hamiltonian cycle (property 2). Further, all periphery nodes are connected by at least one periphery edge (property 3). For example, see Fig. 6a.

Next, assume there exists a shape of $\mathcal{B} = K$ boxes, where K is some integer and $K \geq 1$. Assume the shape has properties 1, 2, and 3. A new valid shape of $\mathcal{B} = K + 1$ boxes could be generated by adding a new box next to an existing box in the $\mathcal{B} = K$ shape such that the two boxes share an entire side. This maintains property 1. The unit path in the newly added box could be merged with one of the periphery edges on the existing path via path merging to maintain property 2 (by Lemma 1).

Further, by creating the new path in this way, property 3 is maintained. This is because the portions of the existing path not involved in the path merge are unchanged and any of the sides of the new box that are on the periphery of the shape have a periphery edge between two periphery nodes. Neither of the edges used in the path merge are on the periphery of the shape, so the $\mathcal{B} = K+1$ path maintains property 3. Examples of this can be seen in each box addition in Fig. 6.

Since all three properties are maintained for the case of $\mathcal{B} = 1$ and for the transition from $\mathcal{B} = K$ to $\mathcal{B} = K + 1$ for some $K \geq 1$, we can conclude by induction that Theorem 1 is valid for any valid shape of size $\mathcal{B} \geq 1$. □

Since we can find a Hamiltonian cycle through a shape by incrementally adding boxes together until we reach that shape, we can conclude that time to find the Hamiltonian cycle would scale with the number of boxes in the shape ($O(\mathcal{B})$ scales linearly with $\mathcal{B}$).

Theorem 1 does not rely on the order in which boxes are assembled to form a shape so long as each additional box results in a valid shape. Furthermore, when a new box is added to an existing shape, it does not matter which periphery edge is used in the path merge. For example, when box 5 is added to the shape in Fig. 6, the periphery edge in box 2 was used to merge with the unit path in box 5. The nearest periphery edges in box 1 or box 4 could have also been used and the result would still have been a planar Hamiltonian cycle.

Finally, because path merging is used to prove Theorem 1 by merging unit paths when each additional box is added to the shape, the resulting planar Hamiltonian cycles have two particular properties. First, each non-spanning edge is directed clockwise around the center of its box because each non-spanning edge originated in a unit path which is clockwise by definition. Second, each spanning edge is part of a pair of parallel and opposite edges that span across the same box connection. In fact, if a tree data representation were used to track the order in which boxes were added to the shape, one would see that these pairs span from parent to child in the tree. For example, box 4 is a child of box 3 when it is added to the shape in Fig. 6e, and a pair of parallel and opposite edges span this parent-child box connection.

We need a term to differentiate planar Hamiltonian cycles that have these two distinct properties from other planar Hamiltonian cycles that do not. In addition, we know from practice that planar Hamiltonian cycles are broken at adjacent start and end nodes to facilitate the movement of robots to and from a charging station. Thus, we define a **valid path** as a planar Hamiltonian cycle with adjacent entry and exit nodes on the periphery of the shape such that each non-spanning edge is directed clockwise around the center of its box and each spanning edge is part of a pair of parallel and opposite edges spanning across the same box connection.

C. Planar Hamiltonian Cycles via the Default Behavior

Earlier we claimed that the default behavior always results in a planar Hamiltonian cycle (for any valid shape). As before, to prove this claim, we have to introduce a new concept: ordered assembly of valid shapes via the **depth-first clockwise-priority (DFCP) method**. Also, we will define a **preferred path** as a planar Hamiltonian cycle that is generated by the default behavior and a **DFCP path** as a planar Hamiltonian cycle that is generated by the DFCP method (and later, we will show that preferred paths and DFCP paths are identical).

We established in Section V-B that a valid shape can be constructed via the assembly of boxes in an arbitrary order. We then constrained the assembly such that each additional box had to maintain a valid shape, and we found that this resulted in valid paths. However, the order in which boxes were assembled remained arbitrary; we only enforced the valid shape constraint. If we restrict the order in which boxes are assembled while enforcing the valid shape constraint, then we can create a subset of valid paths (called DFCP paths) since a sequence of box additions in a particular order is a subset of all box additions in arbitrary orders.

In particular, in the DFCP method, we restrict the order in which boxes are assembled to mimic that of a classical depth-first search where ties are broken in a clockwise-priority manner (thus the name: depth-first clockwise-priority method). Every time a box is added to a shape, we update the frontier of plausible next boxes with un-searched boxes that will be in the desired final shape and that will maintain the properties of a valid shape (i.e., boxes that share a complete side with the most recently added box). Boxes in the frontier are also added to a tree data structure representation of the boxes identified thus far. If more than one box is to box could be added to the tree at the same time, each box is added right to left corresponding to a clockwise order with respect to an imaginary clock face at the center of the most recently added box with the "top" of the clock facing the second most recently added box. For example, boxes 2 and 4 are added to the tree data structure from right to left in Fig. 7b since 2 is more clockwise than 4 with respect to an imaginary clock face centered in box 1 with the "top" of the clock pointing toward box 0. Once all boxes in the frontier have been added to the tree, the deepest, rightmost unexplored node in the tree will be added to the shape next. Once a box is added to the shape, all other instances of it are removed from the frontier. The process continues until there are no other potential boxes to add, and the shape matches the desired final shape (e.g., Fig. 7).

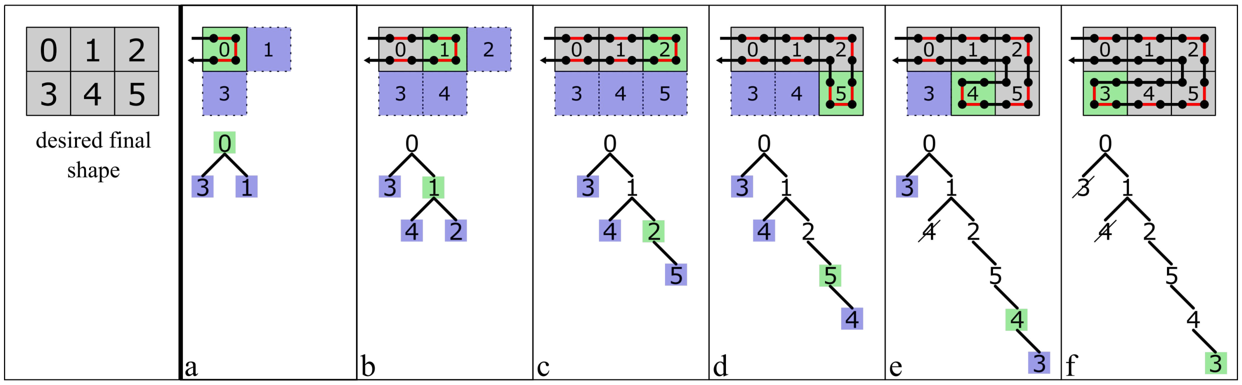

Fig. 7: (left) A desired final shape of 6 boxes and (right) a sequence of temporary shapes and corresponding tree data structures to produce the desired final shape (a-f). Periphery edges are in red. Arrows indicate path direction. Green identifies the most recently added box to the shape, and blue identifies the frontier.

After each addition to the shape, the path is adjusted in the same way as in the proof of Theorem 1: the unit path of the new box is merged with the existing path via path merging. However, unlike before, in the DFCP method we constrain the merge so that the periphery edge of the existing path is in the parent of the new box. For example, when box 3 is added to the shape in Fig. 7f, the periphery edge in box 4 is merged with the unit path of box 3 instead of the periphery edge through box 0 because box 4 is the parent of box 3 in the tree.

The DFCP method results in DFCP paths (e.g., Fig. 7f). As a subset of valid paths, DFCP paths are also planar Hamiltonian cycles with the two unique properties of valid paths: clockwise non-spanning edges and spanning edge pairs between parent and child boxes. The only difference is the order in which the path navigates through boxes. With this established, to prove our claim we only need two additional lemmas about robots moving through a valid shape. Assume we have a robot moving through a shape with some box in that shape, β. Per Algorithm 1, we can say the following about how a robot will choose its next edge, ϵ', with respect to β.

Lemma 3. $\epsilon' = f(x,\beta) = f(x)$ *if* $\forall$ *n in* $\beta : n \notin \mathcal{N}$

Lemma 4. $\epsilon' = f(x,\beta) = f(x)$ *if* $\forall$ *n in* $\beta : n \in V_n$

In layman's terms, Lemma 3 states that finding ϵ' is not a function of a box β until the robot reaches a node adjacent to β and one of the nodes (n) in β is within the list of plausible next nodes $\mathcal{N}$ assembled in Algorithm 1 line 5. Similarly, Lemma 4 suggests that a robot's path is not influenced by the presence of β after the robot has visited all nodes within β since those nodes will be eliminated from the plausible list in Algorithm 1 lines 7 and 8. We will use both lemmas in the proof of Theorem 2.

Theorem 2. *The default behavior always results in a planar Hamiltonian cycle for any valid shape.*

Proof 2. This is a proof by induction that will show that the DFCP path produced by the DFCP method and the preferred path produced by the default behavior are identical for any valid shape as that shape is built up one box at a time.

Consider the base case of the unit box ($\mathcal{B} = 1$). The DFCP path is the unit path with an entry and exit node (e.g., Fig. 7a). Now, consider the preferred path formed by the default behavior. After entering the shape, the robot never sees any new boxes or previously visited boxes (there are none!), so only Algorithm 1 rule 2 applies. The robot moves from node to node in a clockwise manner until it reaches the exit node and exits the shape. The resulting path is identical to the DFCP path.

Next, assume there exists a shape of $\mathcal{B} = K$ boxes, where K is some integer and $K \geq 1$. Assume that the shape has a DFCP path that matches the preferred path generated by the default behavior. One additional box (call it box Y) is added to the shape in accordance with the DFCP method for a total of $\mathcal{B} = K+1$ boxes. A path merge between the existing $\mathcal{B} = K$ path and the unit path of box Y will result in a new DFCP path that passes through box Y via a 5-edge loop between the two nodes nearest to (and in the parent of) box Y: one spanning edge into box Y, three non-spanning edges within box Y, and one spanning edge back into the parent of box Y. No other portions of the preferred path will be changed from the $\mathcal{B} = K$ case. For example, see the addition of box 5 to the shape in Fig. 7d.

A new preferred path generated by the default behavior will match this new DFCP path because the DFCP method and the default behavior explore boxes in the same manner. For the DFCP method, adding box Y to the shape can be thought of as expanding the node in the tree that is the deepest and rightmost unexplored node (where rightmost is analogous to clockwise-priority). Similarly, when a robot is executing the default behavior, it traverses the shape in a depth-first manner due to the priority of Algorithm 1 rule 1 over Algorithm 1 rule 2. When this depth-first precedent is combined with the clockwise priority of Algorithm 1 rule 2, one can conclude that a robot will see and enter neighboring boxes in a depth-first clockwise-priority order.

Since the DFCP method and the default behavior explore boxes in the same order, box Y is both the deepest and rightmost unexplored node in the DFCP method and the last unvisited box the robot reaches when traversing the $\mathcal{B} = K+1$

shape via the default behavior. Further, by Lemma 3 and Lemma 4, the preferred path before the robot enters box Y and after it has visited every node within box Y will be unchanged from the $\mathcal{B} = K$ case. Therefore, we only have to prove that the new DFCP path and the default behavior path are identical within box Y.

Upon reaching box Y, the robot executing the default behavior will follow Algorithm 1 rule 1 to enter box Y. When in box Y, there are no other unvisited boxes, so Algorithm 1 rule 1 will not trigger for the rest of the robot's motion through the shape. The robot will move clockwise in box Y (Algorithm 1 rule 2) and then move into the parent of box Y (Algorithm 1 rule 3). This is the same as the 5-edge loop that the DFCP path takes through box Y back to the parent of box Y.

Since the DFCP path and the preferred path are identical for the case of $\mathcal{B} = 1$ and for the transition from $\mathcal{B} = K$ to $\mathcal{B} = K + 1$ for some $K \geq 1$, we can conclude by induction that the default behavior will always produce a preferred path identical to the DFCP path for any valid shape of $\mathcal{B} \geq 1$. And, since we know DFCP paths are a type of planar Hamiltonian cycle, we know Theorem 2 is valid. Further, we know preferred paths are a subset of valid paths with the two unique properties of valid paths: clockwise non-spanning edges and spanning edge pairs between parent and child boxes. □

VI. Shape Adaptability

Once a swarm has formed a persistent shape in space, the swarm must remain adaptable to shape changes (the addition or removal of a box from the shape). These shape changes can be in arbitrary order and at arbitrary positions around the shape (as long as they maintain a valid shape) and may not be in the same order as the DFCP method. Therefore, robots need a set of behaviors, other than the default behavior, to handle these changes. This section introduces these new behaviors and explains how robots can reconcile shape changes using only local information and making only local path modifications since no single robot can have global influence over the swarm. To do this, we simplify adaptability into three basic steps: 1) detection, 2) primary changes to fill in a new box or empty a removed box, and 3) secondary changes to return the swarm to the preferred path.

To explain these steps, we will use **downstream** to refer to a robot that has previously visited a particular grid node, and **upstream** to refer to a robot that has yet to visit a particular grid node. For example, in Fig. 8a, robots 6 through 11 are all said to be upstream of robot 5's position, and robots 0 through 4 are said to be downstream of robot 5. We will also use **existing shape** and **existing path** to refer to the shape and path that existed prior to the change and **new shape** and **new path** to refer to the post-change shape and post-change path, respectively. The term **interim path** will be used to refer to any provisional path formed in the process of changing from an existing path to a new path.

A. Change Detection

The first step in shape adaptability, detecting the change, is largely outside of the scope of work in this paper because the exact process by which a swarm detects an external environmental stimulus is often dependent on robot hardware. To keep this work hardware agnostic, detection is simplified by using human gestures as environmental stimuli and assuming that robots can sense two unique gestures as the swarm is sculpted: one indicating a box addition and one indicating a box subtraction. Further, we assume that gesture identification is local (only robots nearest to the gesture can sense it), and that a gesture occurs at the location of the desired change. Once a gesture is sensed, robots can propagate notification of the change across the swarm using robot-to-robot communication. It is assumed that a human can initiate a change to any portion of the existing shape that results in a new valid shape. This includes creating and removing holes but excludes changing the entry and exit nodes. It is also assumed that a new change is only initiated after an existing change is completely resolved.

B. Primary Changes

After a shape change has been detected and communicated, the swarm begins making changes to the path. In the case of a box addition, this involves robots filling in the newly added box. In the case of a box subtraction, this involves robots filing out of the newly removed box. These behaviors are captured in Algorithm 2 and described in this section.

In the case of addition, the swarm first identifies the critical point for the change (n_{cp}). This is identified by the robot in the existing shape that has visited only one grid node adjacent to the new box and was planning to take a non-spanning periphery edge as its next move. The grid node occupied by that robot becomes the point of inflection for communicating and addressing the change because all robots downstream of that position may be affected by the change and all robots upstream of that position will not be affected by the change (by Lemma 3). Once n_{cp} is identified, each robot takes action in accordance with Algorithm 2 lines 1-8 depending on whether the robot is upstream, downstream, or at n_{cp}. Once all four nodes in the new box are occupied, the robots have filled the new shape, the interim path is a valid path (Theorem 3 proven later in Section VIII-A), and primary changes are complete.

For example, consider a three-box shape with a preferred path such as the one drawn in Fig. 8a. Assume a fourth box is added to the shape to make a square of four boxes (Fig. 8b). Robot 7, shown in green, is occupying the point of inflection since it has only visited one grid node adjacent to the new box, and, prior to the addition, it intended to take a non-spanning periphery edge for its next move (see the heading of robot 7 in Fig. 8a). With robot 7 at the n_{cp}, the downstream robots (robots 0 through 6) pause their motion until the new box is filled. Meanwhile, robots numbered 7 and up continue to move along the upstream path via the default behavior and fill in the new box via clockwise non-spanning edges (Fig. 8c-f). Once the new box is filled, primary changes are complete and secondary changes can begin.

The case of box subtraction is similar to addition, but there are some key differences, especially with respect to how the existing path is impacted by the box removal. A subtraction

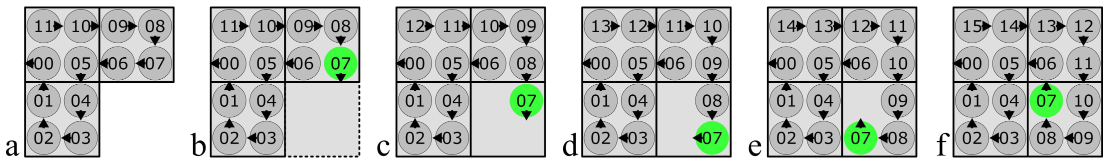

Fig. 8: Primary changes in response to a box added to the shape in (a). Numbers represent robot IDs. Arrows indicate robot heading. Robot 7 (green) is at the point of inflection in (b) and then is the first robot into the new box.

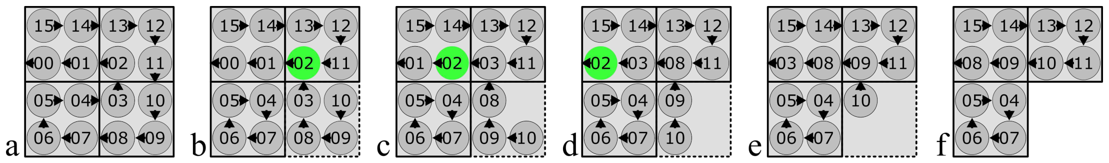

Fig. 9: Primary changes in response to a box subtracted from the shape in (a). Numbers represent robot IDs. Arrows indicate robot heading. Robot 2 (green) is at the point of inflection in (b) and then leads robots 3, 8, 9, and 10 out of the removed box.

Algorithm 2 Primary Changes

```
Input: $CT$, $n_{cp}, V_n$                            ▷ $CT$ = change type
 1: if $CT$ is addition then
 2:     if $n_{cp} \in V_n$ then                      ▷ downstream case
 3:         pause motion and ignore change
 4:     else if robot is at $n_{cp}$ then             ▷ $n_{cp}$ case
 5:         fill in new box via clockwise motion
 6:     else                                          ▷ upstream case
 7:         execute default behavior until reaching $n_{cp}$
 8:     end if
 9: else if $CT$ is subtraction then
        ▷ Set Boolean True if robot is currently
        ▷ within the removed box or False if not
10:     $inRemovedBox \leftarrow$ checkInRemovedBox()
11:     if not $inRemovedBox$ then
12:         if $n_{cp} \in V_n$ then                  ▷ downstream case
13:             execute default behavior
14:         else if robot is at $n_{cp}$ then         ▷ $n_{cp}$ case
15:             execute default behavior
16:         else if $n_{cp} \notin V_n$ then          ▷ upstream case
17:             pause motion
18:             if robot planned to enter removed box then
19:                 plan clockwise non-spanning motion
20:             end if
21:         end if
22:     else
23:         exit box via clockwise motion to $n_{cp}$
24:     end if
25: end if
```

will immediately impact robots in two categories: 1) robots within the removed box and 2) robots that have passed through the removed box but have not yet visited all of the nodes within the removed box. This latter category of robots would have to move through the removed box again to exit the shape via the existing path. Robots that have yet to reach the removed box, and robots that have already visited every node in the removed box, will not be immediately impacted by its removal (by Lemmas 3 and 4, respectively). Therefore, as the change is communicated across the swarm, a point of inflection node (n_{cp}) is identified like before, but now it represents the earliest node in the existing path after which the subtraction will have no effect. This allows robots within the removed box to safely follow the robot at the point of inflection out of the removed box without impacting other portions of the path. The point of inflection is identified as the node occupied by the robot with following criteria:

1) The robot has previously visited all four nodes in the removed box.
2) The robot's previous node was within the removed box.

Once a point of inflection has been identified, each robot takes action in accordance with Algorithm 2 lines 10-24 depending on the robot's relative position to the removed box and n_{cp} (upstream, downstream, etc.). If any of the upstream robots are in a box adjacent to the box to be removed and had planned to move into the box prior to its removal, they adjust their plan to instead move clockwise within their current box. In other words, they adjust their path from a spanning edge to a non-spanning edge. Once all of the robots have exited the box to be removed, primary changes are finished and secondary changes can begin.

For example, consider a shape of four boxes in a 2x2 array with a preferred path cycling robots through the shape (Fig. 9a). Assume the swarm identifies a gesture indicating the removal of the lower right box (Fig. 9b). In such a case, robot 2 would be identified as residing at the point of inflection since it is adjacent to the removed box, it has visited every node in the removed box, and its previous node was within the removed box. With robot 2 at the point of inflection, robots 0, 1, and 2 can continue to move through the shape as if the removal never occurred. Robots upstream of 2's position that are not within the removed box (i.e., robots 4-7 and 11-15) remain still as the box is emptied. Furthermore, robot 4 and robot 11, which had previously planned to take a spanning edges into the removed box, adjusts their paths to take non-spanning edges within their own boxes (Fig. 9a-b). Finally, robots 3, 8, 9, and 10 follow robot 2 out of the removed box until the box is empty (Fig. 9c-f).

Unlike the case of addition, however, the resulting path is not necessarily valid after a subtraction. Primary changes for subtraction can only guarantee that the resulting path is at least pseudo-valid (Theorem 4 proved later in Section VIII-B). Although we will discuss pseudo-valid paths at length in later sections, we introduce the following definition now. Examples of a pseudo-valid paths can be found in Fig. 11b and Fig. 15c.

Definition 2 (Pseudo-valid Path)**.** A discontinuous path comprised of multiple continuous planar closed loops (called sub-cycles) with the following criteria: 1) each node is visited exactly once; 2) each non-spanning edge results in clockwise motion around the center of its box; and 3) each spanning edge belongs to a pair of parallel and opposite spanning edges that cross over the same box connection (though the spanning edges may belong to different sub-cycles).

C. *Secondary Changes*

Secondary changes are the series of local path changes that convert the swarm from following the post-primary-changes path to the preferred path of the new shape. As the last step in shape adaptability, secondary changes are extremely important for shape persistence because they guarantee that the swarm is following the preferred path of the new shape. If the swarm did not return to moving along the preferred path of the new shape after each change, then the impact of each change would have to be communicated to the swarm indefinitely. In those cases, the swarm's path through the shape would forever be altered by the shape changes encountered in the past. By always reverting to the preferred path, the swarm behavior becomes independent of its past, and the swarm can continue to execute the default behavior to persist in the shape while remaining adaptive to future changes.

There are two interchangeable methods for making secondary changes. The first method, called the **communication-based method**, resolves secondary changes by passing a message robot-to-robot through the swarm along the new preferred path immediately after primary changes are complete. Each swarm member reacts to this message and, if necessary, makes a local change to its planned path to match the new preferred path. The swarm then executes the default behavior and moves along the new preferred path as soon as the message has traveled through the shape. The second method, called the **movement-based method**, resolves secondary changes by promoting a single robot to make a series of local path changes as it moves through the remainder of the new shape. This results in a sequence of interim paths until the swarm finally converges to the new preferred path.

The communication-based method is best suited for applications where communication speed is much faster than the traveling speeds of the robots and where secondary changes must be completed quickly (within one time period, τ). This is because secondary changes are complete as soon as the message reaches the robot at the exit node, and one does not have to wait for a robot to travel through the remainder of the new shape (as is the case with the movement-based method). By contrast, the movement-based method is better

Algorithm 3 Memory Message Reception

```
Input: $m, S, V_B, V_n$                         ▷ $m$ = memory message
Output: $m'$                               ▷ $m'$ = new memory message
    ▷ Update data structures with information from $m$
1: $V_B \leftarrow V_B$ from $m$ and $b$                    ▷ $b$ = current box
2: $V_n \leftarrow V_n$ from $m$ and $n$                   ▷ $n$ = current node
    ▷ Determine next edge via default behavior
3: $\epsilon' \leftarrow$ defaultBehavior($b, S, V_B, V_n$)
    ▷ Determine next node
4: $n' \leftarrow$ node at end of $\epsilon'$                  ▷ $n'$ = next node
    ▷ Create message to send to robot at next node
5: $m' \leftarrow$ createMessage($V_B, V_n, n'$)
```

suited for applications where communication speed is slower, and a message is not likely to be passed through the swarm in one time period (τ). In practice, a swarm would be designed to execute one of the two methods for secondary changes for the duration of its task; the swarm would not alternate between the two methods during any given task.

Both methods begin after primary changes have been addressed, and both start from the same grid node in the new shape. For clarity, we define the node from which secondary changes begin as the **secondary change start node (SCSN)**. It represents the latest node in the path where all upstream robots will not be affected by the change (by Lemma 3). In the case of box addition, the point of inflection identified during primary changes (prior to filling in the new box) and the SCSN are identical. In the case of box subtraction, the SCSN is the node occupied by the robot with the following criteria:

1) The robot has never visited any of the nodes in the removed box.
2) Prior to removal, the robot's next position would have been in the removed box.

In the subtraction example, the SCSN is occupied by robot 11 in Fig. 9.

1) Communication-Based Method: In this method, the robot occupying the SCSN initiates a **memory message** through the swarm. The memory message is essentially a virtual robot executing the default behavior along the new preferred path. Instead of physically moving, the message is transmitted from robot to robot along the new preferred path in accordance with Algorithm 3. Each recipient of the memory message re-writes its boxes visited (V_B) and nodes visited (V_n) to match that of the memory message (m). Then, the robot appends its own grid node and box information to the data structures before executing the default behavior without actually moving. By doing this, the robot determines its next position. The robot occupying the original recipient's next position is the next recipient of the updated memory message (m'). The original recipient then transmits the updated memory message to the next recipient and the process continues.

This creates local changes in the path as the message traverses through the swarm. Each recipient adjusts its memory, effectively rewriting its past as if it had always been moving along the new preferred path of the new shape. This ensures that all future robot movements will also be along the new

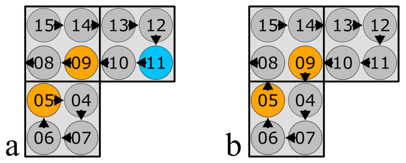

Fig. 10: Secondary changes. (a) The state of the swarm immediately following the primary changes (Fig. 9). Robot 11 is at the SCSN (blue). (b) The state of the swarm after the communication-based method is complete with robots 5 and 9 (orange) having changed their heading. Numbers represent unique IDs of the robots. Arrows indicate robot heading.

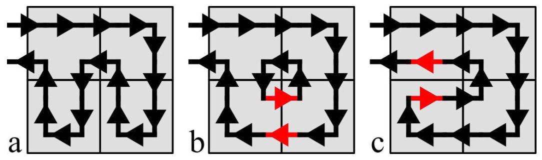

Fig. 11: A sequence of interim paths leading from an interim path (a) to the new preferred path (c). Changes from the previous path are shown in red. Arrows indicate edge direction.

preferred path of the new shape. The memory message is only sent downstream from the SCSN because all upstream portions of the path are the same for both the existing path and the new path (by Lemma 3). Eventually, the memory message reaches the robot at the exit node and the robots resume the default behavior as if nothing ever happened.

Although the communication-based method works for both shape additions and subtractions, we will consider only the example subtraction case (from Fig. 9) for brevity. After the box is removed and primary changes are resolved, robot 11 (at the SCSN) begins transmitting a memory message through the swarm. Fig. 10 shows the state of the swarm when memory message transmission begins (Fig. 10a) and ends (Fig. 10b). Note Fig. 10a is identical to Fig. 9f because secondary changes begin where primary changes left off. In accordance with Algorithm 3, the memory message moves through the swarm along the new preferred path, so it passes through robots in the following order: 11, 10, 9, 4, 7, 6, 5, 8. Robots 11, 10, 4, 7, 6, and 8 do not change their heading direction after processing the memory message because their heading in the interim path (Fig. 10a) is identical to the new preferred path (Fig. 10b). However, robots 9 and 5 both make a local change to their heading in response to the memory message so that the swarm is following the new preferred path (Fig. 10a to b).

2) Movement-Based Method: In this method, secondary changes happen more slowly than in the communication-based method. This is because they rely on a robot moving through the shape, swapping destinations with other robots in the swarm to make local path changes as it goes. This results in a series of interim paths from the path immediately after primary changes complete to the new preferred path. For example, the path after primary changes for the box addition described in Fig. 8 looks like the path drawn in Fig. 11a. This path is changed twice in a sequence of interim paths from Fig. 11a to Fig. 11c in order to achieve the new preferred path (Fig. 11c).

To begin converting the swarm's path to the new preferred path, the swarm promotes a **change robot**. The change robot is not inherently special; it is just the robot that happens to be present at the SCSN when primary changes are completed. As before, all portions of the path upstream of the SCSN are the same for both the existing path and the new preferred path (by Lemma 3), so all impacts of the change robot will come after the change robot's initial position. Thus, the change robot represents the first robot capable of traversing the preferred path of the new shape using only the default behavior.

Even though the change robot is executing the default behavior, robots downstream of its position may not be. Since the post-primary-changes interim path is not necessarily the preferred path (it could be valid or pseudo-valid as proven later for Theorems 3 and 4, respectively), robots downstream of the change robot's position may be executing non-default behavior to maintain a non-preferred interim path (i.e., a pseudo-valid path or a valid path that is not preferred). Non-default behavior is facilitated by **pass-back robots**: robots that use neighbor-to-neighbor communication to "pass back" a non-default movement instructions such that upstream robots follow the pass-back robot's non-default path.

Robots involved in or adjacent to a primary change (i.e., those that fill a new box and those that are adjacent to a newly added or removed box) immediately become pass-back robots when primary changes are finished. Then, after they move to their next grid node, each pass-back robot transmits a **pass-back message** (Algorithm 4) to the robot in its previous grid node (n_{prev}). The recipient of the pass-back message then adjusts its path to follow the pass-back robot (Algorithm 5). The recipient also rewrites its memory (V_B and V_n) to match that of the sender (the original pass-back robot) and then becomes a pass-back robot itself. This ensures that the grid node becomes a semi-permanent instance of non-default behavior because the recipient will pass-back the same message to the next robot in line after it moves. Any upstream robot that receives a pass-back message will follow the robot that previously occupied the grid node and then tell the next upstream robot to do the same.

After transmitting a pass-back message, a pass-back robot converts back to a regular robot and follows the default behavior. However, if it ever receives another pass-back message from a downstream robot, it will become a pass-back robot again (per Algorithm 5).

Pass-back behavior is only cleared from the swarm by the change robot. It does this by moving through downstream grid nodes and ignoring pass-back messages as it moves (i.e., the change robot does not continue to transmit pass-back messages even if a pass-back robot attempts to send a message to the change robot).

Since its downstream neighbors may not be navigating along the same path as the change robot, conflicts can occur when both the change robot and a neighboring robot intend to move to the same node. Such conflicts are resolved via a process called **destination swapping**. In a destination swap, the change robot could take one of two edges from its current node. One edge could be taken in accordance with the default behavior to a node (destination 1), while the other edge

Algorithm 4 Pass-back Robot Behavior: Transmission

```
Input: $\epsilon', V_n, V_B$
Output: $m_{pb}$                              ▷ $m_{pb}$ = pass-back message
1: move to next node via $\epsilon'$
2: $n_{prev} \leftarrow V_n[-1]$                        ▷ get previous node
3: append $b$ to $V_B$     ▷ update boxes visited w. current box
4: append $n$ to $V_n$     ▷ update nodes visited w. current node
   ▷ Create message to send to previous node
5: $m_{pb} \leftarrow$ createMessage($V_B$, $V_n$, $\epsilon'$, $n_{prev}$)
```

Algorithm 5 Pass-back Robot Behavior: Reception

```
Input: $m_{pb}$                               ▷ $m_{pb}$ = pass-back message
1: $V_B \leftarrow V_B$ from $m_{pb}$                  ▷ overwrite boxes visited
2: $V_n \leftarrow V_n$ from $m_{pb}$                  ▷ overwrite nodes visited
3: $\epsilon' \leftarrow \epsilon'$ from $m_{pb}$            ▷ overwrite edge to next node
4: become pass-back robot
```

could be taken in accordance with the current interim path to some other node (destination 2). Since the change robot is always executing its default behavior, it will always take the edge to destination 1. This leaves destination 2 available for the conflicting neighbor robot that had previously planned to move to destination 1 but can no longer since the change robot has priority. Thus, the change robot effectively "swaps destinations" with the conflicting neighbor robot, giving the conflicting neighbor robot destination 2 in exchange for destination 1. In practice, destination swapping is facilitated by the change robot constantly broadcasting its planned movements so that conflicting neighboring robots can react to the message and change their destination before collisions occur.

After destination swapping, the robot that changed its destination to accommodate the change robot becomes a pass-back robot so that other robots continue to follow this new direction and a new interim path is created. This process continues with each destination swap creating a sequence of interim paths until the new preferred path is achieved (Theorem 5 proved later in Section VIII-C). Once the change robot reaches the end of the shape, all pass-back nodes have been cleared, the swarm is executing the new preferred path, and all future robots can continue to execute their default behavior without exception. Finally, the change robot reverts back to a normal swarm robot as it departs for the charging station.

Similar to the communication-based method, the movement-based method transforms interim paths to preferred paths via an identical process for both addition and subtraction. For brevity, only the addition example from Fig. 8 will be discussed here. After addition, the swarm has formed the interim path shown in Fig. 8f. For clarity, a graphic showing the sequence of events for secondary changes is provided in Fig. 12. As before, secondary changes begin where primary changes left off, so Fig. 8f matches Fig. 12a.

An example of pass-back behavior can be seen in the transition from Fig. 12a to 12b. In Fig. 12a, robot 3 is a pass-back robot since it was adjacent to the box addition. When it moves to its next position in Fig. 12b, robot 3 sends a pass-back message to robot 4 and transitions back to the default

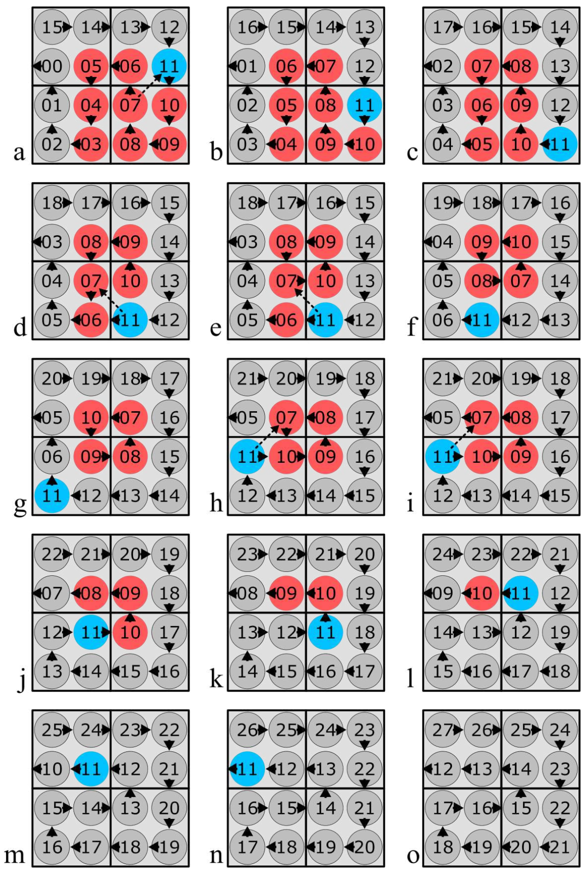

Fig. 12: Secondary changes via the movement-based method. The sequence transforms the interim path (a) to the preferred path (o). Numbers represent robot IDs. Arrows on the robots indicate headings. Dashed arrows indicate communication paths for direction swapping in (d), (e), (h), and (i) and promoting the change robot (a). Red robots are pass-back robots and the blue robot is the change robot.

behavior. Robot 4 receives the pass-back message and adjusts its path to follow robot 3. In Fig. 12a, robot 5 is also a pass-back robot. When it moves to its next position (in Fig. 12b), it sends a pass-back message to robot 6. Robot 5 transitions to the default behavior, but then immediately receives a pass-back message from robot 4, transitioning it back to the pass-back state. All robots involved in the primary change (or adjacent to the primary change) become pass-back robots. They are shown in red in Fig. 12.

Change robot behavior is also evident in Fig. 12. The change robot (robot 11) is promoted by robot 7 once the local changes are complete (Fig. 12a). Robot 11 serves as the change robot in this case because it is the robot present at the SCSN when primary changes are completed. Likewise, robot 7 is the robot that promotes robot 11 because robot 7 was at the point of inflection at the onset of primary changes and was the robot that led the way into the newly added box. Robot 11 then proceeds to move through the shape, executing its default behavior and destination swapping as necessary. For example, in Fig. 12d, robot 11, executing the default behavior, plans to move "left" to the node occupied by robot

6 (see robot 11's heading in Fig. 12d). Thus, robot 7, which had previously planned to move to robot 6's position, is free to swap destinations and instead move "right" to robot 10's position (Fig. 12e). Robot 7 also switches to a pass-back state as a result of the destination swap. This particular destination swap results in a pseudo-valid path of two sub-cycles: robots 7, 8, 9, and 10 form one sub-cycle while robots 3-6 and 11-18 form another. This pseudo-valid path continues to persist until Fig. 12h and Fig. 12i when robots 7 and 11 destination swap again. Lastly, the change robot (robot 11) clears pass-back messages as it moves, so when robot 11 has exited the shape, the swarm is following the new preferred path (Fig. 12o).

VII. Additional Experiments and Demonstrations

We performed additional experiments to demonstrate the swarm's adaptability while maintaining its persistence. Specifically, we show detection, primary changes, and secondary changes via both the communication-based method and the movement-based method with the swarm executing the default behavior in between shape changes. These demonstrations were completed on a swarm of mobile robots to capture the swarm's response to an actual human and on a simulated swarm of 90 robots to depict the scalability of the algorithms.

With no known direct state-of-the-art comparisons, it was not possible to compare the performance of these demonstrations against the metrics of other methods. Instead, these demonstrations were completed as a proof of concept to both reinforce the effectiveness of the algorithms and complement the theoretical proofs. Furthermore, although performance reliability was not a primary driver of our work, we did note that the simulated experiments could be run repeatedly without error. The mobile robot experiments were also repeatable with exceptions only in the event of unrelated robot hardware issues (e.g., robot wheels getting snagged on hairs on the floor, etc.).

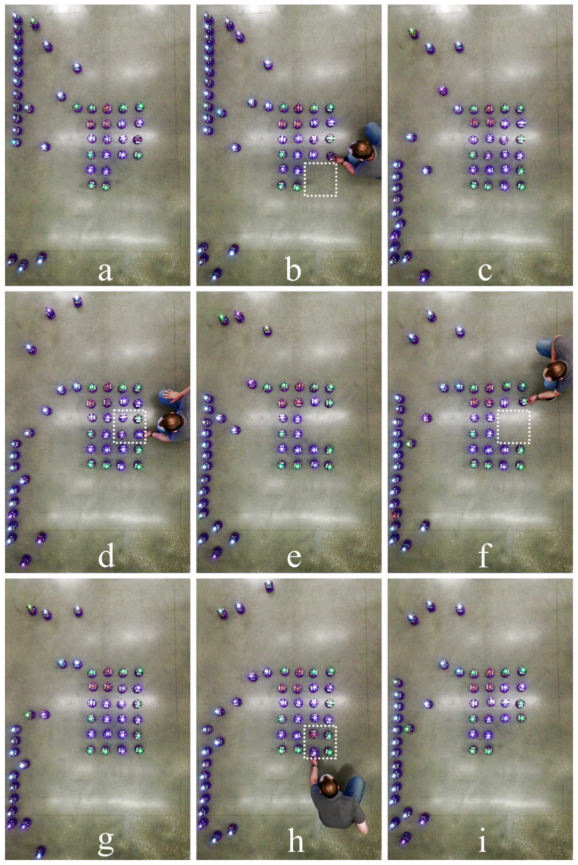

Fig. 13: A sequence of images from a *Coachbot* experiment running the movement-based method. Pane (a) shows the initial shape. Human initiated additions are in panes (b) and (f), and subtractions are in panes (d) and (h). A dashed rectangle indicates the added or removed box. Panes (c), (e), (g), and (i) show the swarm's shape once the previous change is resolved. In-shape robots have green, blue, and red LEDs to indicate the potential for addition, subtraction, and no change, respectively.

A. Demonstrations with Humans

Experiments were performed on the swarm of *Coachbots* to show the swarm's response to a human sculpting the shape. These experiments used a 20 robot initial shape and a 22 robot charging station. The demonstrations on the *Coachbots* also consisted of both addition and subtraction via both secondary change methods (communication-based and movement-based).

In these experiments, once the initial shape was formed and the robots reached a steady-state persistent cycle into and out of the shape, a human initiated a random set of arbitrary changes to the shape by interacting directly with the swarm. In both secondary change methods, and for both addition and subtraction, the swarm effectively detected each change, made primary changes, and finally made secondary changes to continue to cycle robots through the new shape. The time to resolve each change was on the order of 1-7 minutes depending on the location of the change. The changes made closer to the end of the path took less time to resolve than those towards the beginning where the memory message or change robot would have to travel further before the change completed. These experiments were recorded with an overhead camera, and images of one human-swarm interaction experiment are captured in Fig. 13.

The *Coachbots* were used in these experiments because they were an existing swarm available to the authors. Unfortunately, they have no means of sensing humans: only their position (x, y) and heading (θ). Therefore, we physically rotated robots in place as human "gestures" when sculpting the shape. Robots changed color within the shape to indicate which robots could be manipulated for an addition and which for a subtraction. Robots would turn green if they were along the periphery of the shape alongside a box that could be added to the shape. Robots would turn blue if they were not along the periphery of the shape or otherwise not alongside a box that could be added to the shape. Finally, robots would turn red if they could not be manipulated for either addition or subtraction (e.g., a robot at the exit node). If a green robot's heading was changed, then the robot would initiate a box addition, and if a blue robot's heading was changed, then the robot would initiate a box subtraction. Red robots were unresponsive to human interaction. Other robot colors (e.g., yellow and white) are

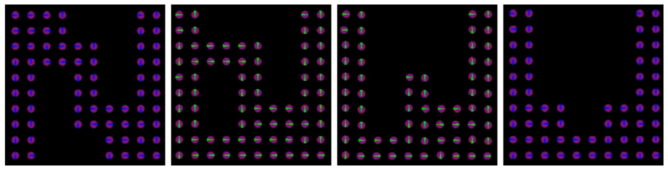
Fig. 14: A simulation of 90 robots running the communication-based method. Robots begin in a 17-box “N” (far left). Boxes are removed and added to the shape (left to right) until the swarm forms the 15-box “U” (far right). Robot LED color indicates shapes of with an even (green) or odd (blue) number of boxes. The charging station is not shown.

used to debug the system.

B. Large Scale Simulations

The final set of experiments demonstrated the scalability of the algorithms by simulating a swarm of 90 robots forming a 17-box “N” shape persistently. The swarm was then manipulated into a 15-box “U” shape through a series of additions and subtractions. Human gestures indicating additions or subtractions were emulated by pre-programmed messages transmitted at pre-assigned times to the robots nearest to the desired addition or subtraction. Images of a particular test are in Fig. 14. This simulation demonstrates the scalability of the algorithms to swarms of large numbers and shows the complexity of the shapes that can be formed despite the valid shape constraints. Although scalability is not formally proven, the size of the swarm is not a necessary component in any of the algorithms, and each robot is executing identical behavior without a dependency on a centralized actor. This indicates that the size of the swarm would not be limited by the algorithms, but rather the capacity of the charging station, the size of the path that an individual robot can travel, the onboard memory of each robot, or the size of the environment.

VIII. Shape Adaptability Theory

The previous sections described swarm adaptability in the cases of box addition and box subtraction. Specifically, we covered the steps of detection, primary changes (for both addition and subtraction), and secondary changes (for both the communication-based method and the movement-based method). Along the way, we stated a series of unproven claims as fact. The remainder of this section formally addresses these claims and provides proofs to support their validity.

A. Primary Changes: Addition Yields Valid Paths

Earlier we claimed that, in the event of an addition, primary changes in accordance with Algorithm 2 lines 1-8 always result in a valid interim path. We can prove this claim directly by the logic of Theorem 1.

Theorem 3. *Addition primary changes result in a valid path.*

Proof 3. Prior to an addition, the existing shape is a valid shape and the existing path is a preferred path. In the event of an addition, primary changes effectively replace a periphery edge of the existing path with a 5-edge loop: 1 edge that spans into the new box from an existing box, 3 non-spanning edges that move robots clockwise through the new box, and 1 spanning edge back into the existing box. This is the same 5-edge loop discussed in the proof of Theorem 1 that occurs when a unit path through a new box is merged into the existing path. Thus, by the same logic, we know that the 5-edge loop created by primary changes in response to an addition will also result a valid path and Theorem 3 is valid. □

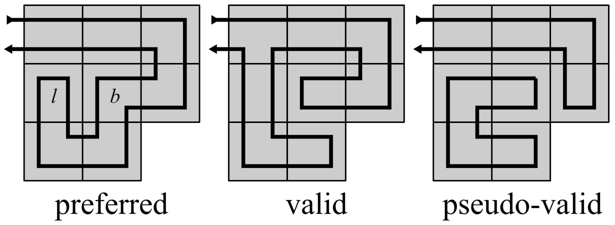

Fig. 15: Examples of preferred, valid, and pseudo-valid paths through the same shape. Arrows indicate path direction.

B. Primary Changes: Subtraction Yields Pseudo-valid Paths

Earlier we claimed that local changes in response to a subtraction in accordance with Algorithm 2 lines 9-25 can only guarantee that the resulting path is at least pseudo-valid. In order to prove this, we must explain the phrase “*at least pseudo-valid*.” We will also establish two more lemmas.

Lemma 5. *For a path that is pseudo-valid, valid, or preferred, each non-spanning edge results in clockwise motion around the center of its box.*

Lemma 6. *For a path that is pseudo-valid, valid, or preferred, each spanning edge belongs to a pair of parallel and opposite spanning edges that cross over the same box connection.*

By Definition 2, in a pseudo-valid path, each non-spanning edge results in clockwise motion around the center of its box, and each spanning edge belongs to a pair of parallel and opposite spanning edges that cross over the same box connection. Since these properties are also true of preferred and valid paths, we can conclude Lemma 5 and Lemma 6. Examples of a preferred path, valid path, and pseudo-valid path are shown in Fig. 15 for comparison.

Since preferred paths are a subset of valid paths, and valid paths are a subset of pseudo-valid paths (i.e., valid paths are pseudo-valid paths with only one sub-cycle), then we can use the phrase “*at least a pseudo-valid path*” to refer to any path that is pseudo-valid but could also be valid or preferred. With these terms defined, we can now prove our claim directly using the concepts of path separation, path merging, and Lemma 2.

Theorem 4. *Subtraction primary changes result in at least a pseudo-valid path.*

Proof 4. Prior to a subtraction, the existing shape is a valid shape and the existing path is a preferred path. Since the existing path is a preferred path, by Theorem 2 we know that a robot traversing the preferred path will create a tree data structure of boxes that matches the tree data structure assembled by the DFCP method (just like in Fig. 7). The box with entry and exit nodes is the root of the tree, and the remainder of the boxes in the shape are either branches or leaves. Branches are boxes that the robot enters and exits at least twice: at least once to access descendent boxes and once to move back up the tree to exit the shape. On the other hand, leaves are boxes for which the robot enters and exits only once, making a distinctive 5-edge "∪" pattern through the box. For example, in Fig. 15, the box marked "l" is a leaf and the box marked "b" is a branch. Now, let R denote a box to be removed. The impact of removing R from a valid shape (with a preferred path) varies depending on whether R is a leaf or a branch.

If R is a leaf, then the box can be removed without impacting any other portions of the path because the original path did not need to travel through R to get to other boxes. When R is removed, the robot that had previously planned to take a spanning edge into R will instead plan to move clockwise within its own box (in accordance with primary change procedures). In that case, removing R is like separating its unit path from the planar Hamiltonian cycle in the remainder of the shape via path separation (i.e., spanning edges become non-spanning edges). This results in a planar Hamiltonian cycle through the remainder of the shape (by Lemma 2), and the path prior to and after R is not affected by the presence of R (by Lemmas 3 and 4). Thus, we can conclude that the planar Hamiltonian cycle that remains once R is removed is identical to the planar Hamiltonian cycle that would have existed if R had never been a part of the shape. We can also conclude that the planar Hamiltonian cycle that would have existed if R had never been a part of the shape is the same as the preferred path that a robot would have traveled via the default behavior if R had never been a part of the shape as R (being a leaf) is the last unvisited box in its subtree. In other words, we can conclude that when a leaf box is removed, primary changes result in the preferred path of the new shape.

However, if R is a branch, then removing the box will result in a discontinuity in the path since the path had to pass through R to get to other boxes. In that case, all robots that had previously planned to take a spanning edge into R will instead plan to move clockwise within their own boxes (in accordance with primary change procedures). This is the same as a path separation operation on each pair of spanning edges that crossed a side of R. By Lemma 2, this still creates planar Hamiltonian cycles in the remainder of the boxes, but instead of being one continuous path, the resulting path will have at least two sub-cycles. For example, if R is a branch with only one child, there will be one sub-cycle through all of the boxes visited from the root box through the parent of R and one sub-cycle through all of boxes visited after R. By Definition 2, such a path is a pseudo-valid path, so we can conclude that when a branch is removed, primary changes result in a pseudo-valid interim path.

Finally, since a primary changes in response to a subtraction result in either a preferred path (in the case of a leaf) or a pseudo-valid path (in the case of a branch), we can conclude that Theorem 4 is valid. □

C. Secondary Changes Yield Preferred Paths

In describing secondary changes, we presented two interchangeable methods for operation: the communication-based method and the movement-based method. We also explained that secondary changes return the swarm to the preferred path of the new shape so that robots can freely execute the default behavior without concerning themselves about shape changes that occurred in the past. Therefore, we must prove that both the communication-based method and the movement-based method result in the new preferred path of the new shape. The proof of the communication-based method is trivial. Since the communication-based method is just a virtual implementation of the default behavior, we can conclude that it will always result in the new preferred path of the new shape by Theorem 2.

However, the proof of the movement-based method is less obvious. We earlier claimed that by repeatedly destination swapping with the change robot, the swarm will morph through a sequence of interim paths until the preferred path is achieved. This relies on two basic tenets: 1) that a destination swap morphs the swarm from one path that is at least pseudo-valid to another path that is at least pseudo-valid, and 2) that the sequence of interim paths will converge to the preferred path. Both of these are addressed in the proof of Theorem 5. We will also use the following lemmas.

Lemma 7. *For a robot at a node in at least a pseudo-valid path, only one outgoing non-spanning edge exists for the robot to travel within its box while maintaining Lemma 5.*

Lemma 8. *For a robot at a node in at least a pseudo-valid path, only one outgoing spanning edge exists for the robot to travel into a different box while maintaining Lemma 5 and Lemma 6.*

Lemma 7 is trivial based on the definitions of preferred, valid, and pseudo-valid paths, and Fig. 4a shows each of the four valid outgoing spanning edges for each of the four nodes in a unit box. On the other hand, Lemma 8 is slightly less intuitive, so we provide Fig. 16 to show a visual representation of Lemma 8 for one node in an arbitrary box in an arbitrary valid shape (the argument is rotationally symmetric for all other nodes in the box). Of the two possible options for spanning edges leading away from the white node in Fig. 16, only edge a is compatible for paths that are at least pseudo-valid. If edge b were present, then edge c must also exist by Lemma 6. However, the only way a path could connect from the end of edge b to the start of edge c without intersection is for the path to have a non-spanning edge with *counter-clockwise* motion around the center of its box (if not in the box that edge b spans into then in some other leaf box downstream of edge b). Since this contradicts Lemma 5, we can infer that edge b will never exist in a pseudo-valid, valid, or preferred path and Lemma 8 must be valid.

Theorem 5. *In the movement-based method of secondary changes, a sequence of valid and pseudo-valid interim paths created as a change robot moves through the shape will converge to the preferred path of the new shape.*

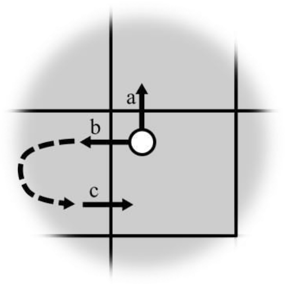

Fig. 16: Edge a is the only plausible spanning edge to leave the white node in a path that is at least pseudo-valid. Edges b and c cannot exist without a counter-clockwise spanning edge somewhere in the shape (indicated by the dashed line).

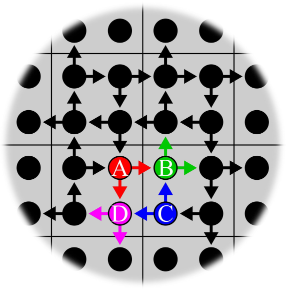

Fig. 17: Each valid edge for nodes in an arbitrary shape.

Proof 5. First, by Theorem 3 and Theorem 4, it can be said that all local changes, regardless of whether a box was added or removed, will result in at least a pseudo-valid path (i.e., the path is either pseudo-valid, valid, or preferred). Thus, it can be assumed that when a change robot is promoted at the start of secondary changes, the interim path is at least pseudo-valid, so we have to prove that a destination swap results in another path that is at least pseudo-valid.

To do this, we use Lemma 7 and Lemma 8, to draw the two potential outgoing edges at each node in an arbitrary shape in Fig. 17. Consider a change robot at node A (the argument is rotationally symmetric for all other nodes in a box). Since we know a destination swap will only occur when a neighboring robot is planning to move to the same destination as the change robot, then we can see only two cases where a destination swap might occur: 1) when the change robot is planning to take edge $\overrightarrow{AB}$ at the robot at node C is planning to take edge $\overrightarrow{CB}$, or 2) when the change robot is planning to take edge $\overrightarrow{AD}$ at the robot at node C is planning to take edge $\overrightarrow{CD}$. In both cases, the robot at node C is free to swap destinations without risking a collision with other robots because the node it is swapping to can only be accessed by one of two edges, and the other edge is not being taken by the change robot. For example, in the first case, the robot at node C is free to swap its destination from node B to node D because the change robot is going to take edge $\overrightarrow{AB}$ (not $\overrightarrow{AD}$), meaning no other robot is planning to move to node D.

Furthermore, all aspects of valid and pseudo-valid paths are maintained after these destination swaps because, in both cases, there are no path intersections, each node is still connected via exactly two edges, the start and end nodes have not been changed, and no periphery edges have been changed. The only impact of swapping destinations is that the nodes may belong to a different sub-cycle (or valid path) than they previously belonged to. Thus, we can conclude that the path remains at least pseudo-valid after a destination swap.

Finally, as the change robot moves, it creates a sequence of interim paths as it destination swaps with its neighbors. It always executes the default behavior, and so do all of the robots upstream of its position. This implies that, as the change robot moves closer to the end of the shape, more robots are following the new preferred path via the default behavior and fewer robots are following the interim path. Therefore, a sequence of valid and pseudo-valid interim paths created as a change robot moves through the shape will converge to the new preferred path as the change robot converges toward the exit node, and Theorem 5 is valid. □

IX. Conclusion

In this work, we presented algorithms for persistent and adaptive 2D shape formation that allow a swarm to overcome the limitations of individual robot power constraints while finding new paths through the shape after every shape change. We have shown that these algorithms are provably correct and have demonstrated their effectiveness in both simulation and a swarm of physical robots. More significantly, though, we have opened the door to a largely unexplored field of swarm shape formation applications where the duration of the task may no longer be a constraint on the swarm's performance and the shape that the swarm is forming during its task is free to change in response to an external stimulus such as a human sculpting the shape.

Future work includes demonstrating these algorithms on a swarm of physical flying robots. There is also work to be done to improve the efficiency and fault tolerance of these algorithms to make them more suitable for applications in unpredictable environments. Specifically, the behavior of the swarm in the face of challenges, such as communication interference and damaged or adversarial robots, has not yet been explored. Investigating such impacts on the algorithms would also require us to identify potential single points of failure and build redundancy into the algorithms (e.g, detecting a failure in the transmission of a memory message and recovering by restarting transmission of a new message).

Finally, we intend to develop three-dimensional algorithms that are analogous to the 2D algorithms presented in this paper so that swarm implementations are not constrained to planar tasks. One way to accomplish this might be to execute 2D algorithms for a set of vertically stacked layers in a 3D shape. Another method might be to develop valid 3D shapes constructed of cubes in the same way we developed valid shapes constructed of boxes in 2D. Additional challenges with a 3D version will include the impact of downwash on other flying robots, the ability to detect human gestures while in flight, and the safety concerns associated with a human interacting with flying robots.

(REVISION NOTE: References [11], [12], [22]–[27] are new.)

References

[1] A. G. Curtis *et al.*, "Autonomous 3d position control for a safe single motor micro aerial vehicle," *IEEE RA-L*, 2023.

[2] H. Wang and M. Rubenstein, "Shape formation in homogeneous swarms using local task swapping," *IEEE T-RO*, vol. 36, pp. 597–612, 3 Jun. 2020.

[3] H. Xu, H. Guan, A. Liang, and X. Yan, "A multi-robot pattern formation algorithm based on distributed swarm intelligence," in *IEEE ICCSEA*, vol. 1, 2010, pp. 71–75.

[4] Z. Xue and J. Zeng, "Formation control numerical simulations of geometric patterns for unmanned autonomous vehicles with swarm dynamical methodologies," in *IEEE International Conference on Measuring Technology and Mechatronics Automation*, 2009.

[5] J. Wu *et al.*, "Distributed uav swarm formation and collision avoidance strategies over fixed and switching topologies," *IEEE Transactions on Cybernetics*, 2021.

[6] Y. Liu *et al.*, "A distributed control approach to formation balancing and maneuvering of multiple multirotor uavs," *IEEE T-RO*, 2018.

[7] M. Aranda *et al.*, "Distributed formation stabilization using relative position measurements in local coordinates," *IEEE Transactions on Automatic Control*, 2016.

[8] H. Wang and M. Rubenstein, "Generating goal configurations for scalable shape formation in robotic swarms," in *DARS*, Springer, 2021.

[9] X. Dong *et al.*, "Time-varying formation control for unmanned aerial vehicles: Theories and applications," *IEEE Transactions on Control Systems Technology*, 2014.

[10] Y. Xu *et al.*, "Concurrent optimal trajectory planning for indoor quadrotor formation switching," *Journal of Intelligent & Robotic Systems*, 2019.

[11] G. A. Di Luna *et al.*, "Shape formation by programmable particles," *Distributed Computing*, 2020.

[12] Z. Derakhshandeh *et al.*, "Universal shape formation for programmable matter," in *SPAA*, 2016.

[13] T. M. Cabreira, L. B. Brisolara, and R. F. Paulo, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, pp. 1–38, 1 Mar. 2019.

[14] Y.-S. Jiao *et al.*, "Research on the coverage path planning of uavs for polygon areas," in *IEEE ICIEA*, 2010.

[15] W. H. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *IEEE ICRA*, 2001.

[16] M. Torres *et al.*, "Coverage path planning with unmanned aerial vehicles for 3d terrain reconstruction," *Expert Systems with Applications*, vol. 55, pp. 441–451, 2016.

[17] C. Di Franco and G. Buttazzo, "Coverage path planning for uavs photogrammetry with energy and resolution constraints," *Journal of Intelligent & Robotic Systems*, 2016.

[18] T. M. Cabreira *et al.*, "Energy-aware spiral coverage path planning for uav photogrammetric applications," *IEEE RA-L*, 2018.

[19] D. Albani, D. Nardi, and V. Trianni, *Field Coverage and Weed Mapping by UAV Swarms*. 2017.

[20] S.-H. Lim and H.-C. Bang, "Waypoint planning algorithm using cost functions for surveillance," *IJASS*, 2010.

[21] J. Zelenka and T. Kasanickỳ, "Insect pheromone strategy for the robots coordination—reaction on loss communication," in *IEEE CINTI*, 2014.

[22] F. Arvin *et al.*, "Perpetual robot swarm: Long-term autonomy of mobile robots using on-the-fly inductive charging," *Intelligent & Robotic Systems*, 2018.

[23] M. Carrillo *et al.*, "A bio-inspired approach for collaborative exploration with mobile battery recharging in swarm robotics," in *BIOMA*, Springer, 2018.

[24] M. Rappaport and C. Bettstetter, "Coordinated recharging of mobile robots during exploration," in *IEEE IROS*, 2017.

[25] G. Li, I. Svogor, and G. Beltrame, "Long-term pattern formation and maintenance for battery-powered robots," *Swarm Intelligence*, 2019.

[26] A. Boggio-Dandry and T. Soyata, "Perpetual flight for uav drone swarms using continuous energy replenishment," in *IEEE UEMCON*, 2018.

[27] D. Mitchell, E. A. Cappo, and N. Michael, "Persistent robot formation flight via online substitution," in *IEEE IROS*, 2016.

[28] M. R. Garey, D. S. Johnson, and R. E. Tarjan, "The planar hamiltonian circuit problem is np-complete," *SIAM Journal on Computing*, 1976.

[29] C. Umans and W. Lenhart, "Hamiltonian cycles in solid grid graphs," in *Proceedings 38th Annual Symposium on Foundations of Computer Science*, 1997.

[30] R. I. Nishat, "Reconfiguration of hamiltonian cycles and paths in grid graphs," Ph.D. dissertation, 2020.

[31] K. C. Cheung *et al.*, "Programmable assembly with universally foldable strings (moteins)," *IEEE T-RO*, 2011.

[32] A. A. Joshi, M. C. Bhatt, and A. Sinha, "Modification of hilbert's space-filling curve to avoid obstacles: A robotic path-planning strategy," in *2019 Sixth Indian Control Conference*, IEEE.

[33] S. A. Sadat, J. Wawerla, and R. Vaughan, "Fractal trajectories for online non-uniform aerial coverage," in *IEEE ICRA*, 2015.

[34] A. Kolling *et al.*, "Human interaction with robot swarms: A survey," *IEEE Transactions on Human-Machine Systems*, 2015.

[35] A. Giusti *et al.*, "Human-swarm interaction through distributed cooperative gesture recognition," in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, 2012.

[36] J. Nagi *et al.*, "Human-swarm interaction using spatial gestures," in *IEEE IROS*, 2014.

Appendix

A. *Defined Terms*

Term		Definition
box	:	four grid nodes in a 2x2 square
change robot	:	the robot whose movement initiates destination swapping in the movement-based method
communication-based method	:	a method for secondary changes that converts an interim path to the new preferred path by passing a memory message
default behavior	:	algorithm used for shape persistence; results in a preferred path
depth-first clockwise-priority (DFCP) method	:	a method for assembling valid shapes by assembling boxes in accordance with a traditional depth-first search where ties are resolved with a clockwise priority
destination swap	:	process by which a neighboring robot assumes the interim path destination of the change robot instead of its own interim path destination
DFCP path	:	a type of valid path generated by the DFCP method
downstream	:	refers to a robot that has previously visited a particular grid node
existing path	:	the path prior to a change (i.e., addition or subtraction)
existing shape	:	the shape prior to a change (i.e., addition or subtraction)
interim path	:	a provisional path formed in the process of changing from an existing path to a new path
memory message	:	the message used to communicate change in the communication-based method
movement-based method	:	a method for secondary changes that converts an interim path to the new preferred path by a series of destination swaps initiated by the movement of a change robot
new path	:	the path after a change is resolved
new shape	:	the shape after a change is resolved
non-spanning edge	:	an edge that begins and terminates in the same box
opposite edges	:	edges that have directions exactly 180° from one another
pair	:	(e.g., a *pair* of edges) two edges with start and end nodes in a 2x2 grid configuration
parallel edges	:	edges that are equidistant everywhere and do not intersect
pass-back message	:	message used to communicate non-default movement during secondary changes
pass-back robots	:	robots that send pass-back messages to upstream robots
path merging	:	the process of replacing a pair of parallel and opposite non-spanning edges with a pair of parallel and opposite spanning edges
path separation	:	the process of replacing a pair of parallel and opposite spanning edges with a pair of parallel and opposite non-spanning edges
periphery edge	:	a non-spanning edge between two periphery nodes
periphery node	:	an in-shape node along the periphery of the shape
preferred path	:	a type of valid path generated by the default behavior
pseudo-valid path	:	a discontinuous path comprised sub-cycles such that: 1) each node is visited exactly once; 2) each non-spanning edge results in clockwise motion around the center of its box; and 3) each spanning edge belongs to a pair of parallel and opposite spanning edges that cross over the same box connection (though they may belong to different sub-cycles).
secondary change start node (SCSN)	:	the node from which secondary changes begin
spanning edge	:	an edge that begins and terminates in different boxes
sub-cycle	:	a planar Hamiltonian cycle through a portion of the shape
unit path	:	a clockwise Hamiltonian cycle through a unit shape
unit shape	:	a shape constructed of 1 box
upstream	:	refers to a robot that has yet to visit a particular grid node
valid path	:	a planar Hamiltonian cycle with adjacent entry and exit nodes on the periphery of the shape such that each non-spanning edge is directed clockwise around the center of its box and each spanning edge is part of a pair of parallel and opposite edges spanning across the same box connection
valid shape	:	a shape constructed of boxes assembled continuously such that each box meets the full side of another box