Vision-Based Self-Assembly for Modular Multirotor Structures

Yehonathan Litman, Neeraj Gandhi, Linh Thi Xuan Phan, David Saldaña

***Abstract*— Modular aerial robots can adapt their shape to suit a wide range of tasks, but developing efficient self-reconfiguration algorithms is still a challenge. Self-reconfiguration algorithms in the literature rely on high-accuracy global positioning systems which are not realistic for real-world applications. In this paper, we study self-reconfiguration algorithms using a combination of low-accuracy global positioning systems (e.g., GPS) and on-board relative positioning (e.g. visual sensing) for precise docking actions. We present three algorithms: 1) parallelized self-assembly sequencing that minimizes the number of serial "docking steps"; 2) parallelized self-assembly sequencing that minimizes total distance traveled by modules; and 3) parallelized self-reconfiguration that breaks an initial structure down as little as possible before assembling a new structure. The algorithms take into account the constraints of the local sensors and use heuristics to provide a computationally efficient solution for the combinatorial problem. Our evaluation in 2-D and 3-D simulations show that the algorithms scale with the number of modules and structure shape.**

I. Introduction

The design of aerial vehicles for applications like rapid infrastructure construction [18], [19], [23], load transportation [4], [13], cargo lifting [3], [16], [18] and search and rescue [12], [15], [24], requires considering a fundamental trade-off between agility and strength. For instance, a large robot is well-suited to lifting humans in a search-and-rescue scenario, but could find it difficult to navigate through wreckage. Small robots are more agile, but their payload capacity is very limited, often insufficient to lift a human. With traditional robotic systems, we would have to pick the larger robot to ensure that humans can be rescued. Modular robots provide a promising alternative that can be both powerful and agile since the small modules can individually navigate to a location before interlocking to form a structure powerful enough to lift a person.

An aerial system composed of multiple modules creates a versatile "meta" robot that can adapt to task requirements by self-reconfiguration. Self-reconfiguration is composed of two steps: *i)* finding an efficient sequence of steps to reconfigure the robot, and *ii)* performing docking and undocking actions to create and break physical links. The last part requires a precise maneuver to align the vehicles in midair and create the link. Although there is existing work in the self-assembly of aerial robots [5], [16], [18], [24], these typically rely on high-accuracy global positioning systems. Other platforms consider positioning systems like the Global Positioning System (GPS) [7], [10], but GPS accuracy is not high enough for aerial docking (i.e., millimeter accuracy). There has also been some work on ground- and water-based modular robots that use vision to determine the relative location of modules for docking [6], [9], [12]. In aerial robotics, vision has been used for perching [21], [22] and docking [11].

Manuscript received: October 15th, 2020; Revised January 11th, 2021; Accepted February 3rd, 2021.

This paper was recommended for publication by Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers' comments.

Yehonathan Litman is with the Robotics Institute, Carnegie Mellon University: litman@cmu.edu. Neeraj Gandhi and Linh Thi Xuan Phan are with the PRECISE Center and Distributed Systems Laboratory, University of Pennsylvania: {ngandhi3, linhphan}@seas.upenn.edu. David Saldaña is with the Autonomous and Intelligent Robotics Laboratory (AIRLab), Lehigh University: saldana@lehigh.edu.

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

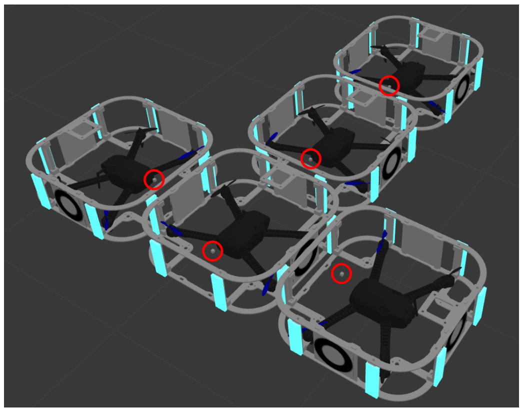
Fig. 1: Five modules assembling a T-structure, relying only on local vision. The white points highlighted with a red circle represent the cameras. Magnets are turquoise rectangles and WhyCon tags [14] are hollow black circles.

We propose a hybrid approach by combining low-accurate global positioning information for determining the docking sequence and local perception for high-accuracy docking actions. Unlike previous approaches [5], [11], [19], this approach considers constraints like formation deadlocks that result from onboard sensors. For instance, if two single-camera modules faced one another, they would be unable to dock to other modules because each would block the other's field of view, rendering the full structure infeasible.

Our approach focuses on modular quadrotor aerial robots equipped with cameras, but extends to other environments (ground, aquatic), shapes (convex, symmetric, polygon-shaped modules), and relative positioning sensors (e.g., LIDAR). Docking actions are parallelized as much as possible, so our techniques are efficient with respect to the number of "serial" docking steps. Further, the techniques scale with the number of modules and work across structure shapes. In sum, we make three main contributions: 1) a self-assembly algorithm that is efficient in number of serial "docking actions", 2) a self-assembly algorithm that minimizes the sum distance all robots must travel, and 3) a self-reconfiguration algorithm that finds an optimal plan to transform a structure from one shape into another.

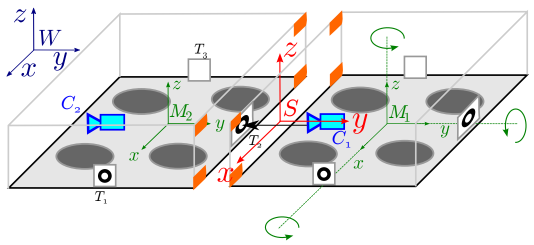

Fig. 2: Two modules docking to form structure $\mathcal{S}$. Module M_1 aligns its camera, C_1, with the y-axis and tag T_2 to dock to the back side of M_2.

II. ModQuad-Vi Model

Our modular aerial system is propelled by autonomous modules based on [11]. A module is defined as follows.

Definition 1. (Module) *A module, M, is a quadrotor-propelled cuboid that can horizontally dock to other modules. The cuboid has length and width w and height h.*

Modules are homogeneous; they have identical mass m, dimensions, inertia tensor, sensors and actuators. Each vertical face of the cuboid frame has four magnets (one magnet per corner). The left, right, and back sides of the cuboid have WhyCon tags [14]. Each module has a front-facing camera that is used for onboard tag detection [14]. An individual tag's pose relative to a camera can be detected at high frequency ($>$ 30 Hz). A structure is a set of attached modules forming a rigid body. Formally, it is defined as:

Definition 2. (Structure) *A structure, or robot, $\mathcal{S} = \{M_1, ..., M_N\}$, is a non-empty set of rigidly connected modules that behave as a single rigid body. Docking is always in a horizontal direction; thus, structure height is always h.*

A pair of modules can dock in midair, whereby one module hovers in place and the other tracks a WhyCon tag on the first to guide itself (and other attached modules) to attach to the tracked tag-face of the hovering module. These are *waiting* (M_w) and *docking* (M_d) modules, respectively.

The inertial frame, W, is fixed and has its z-axis pointing upwards. The center of mass of the i^{th} module M_i in the inertial frame is denoted $\mathbf{x}_i \in \mathbb{R}^3$. The origin of the structure coordinate frame, S, is at the structure's center of mass. The graph $\mathcal{G} = \{\mathcal{S}, \mathcal{E}\}$ is the set of modules, $\mathcal{S}$, and the set of inter-module connections, $\mathcal{E} = \{||\mathbf{x}_i - \mathbf{x}_j|| = w, \forall (M_i, M_j) \in \mathcal{S}\}$. We assume that we directly control accelerations, in addition to linear and angular velocities. Please refer to [18] for more dynamics details.

Docking is a directed operation: one module's camera must point to another module's tag. Fig. 2 shows a docking (M_1) and a waiting (M_2) module. Eight permanent magnets, shown as orange rectangles in the figure, magnetically connect the modules. The gray ellipses represent the rotors, and the blue, red, and green axes represent the world, structure, and module coordinate frames, respectively. The docking module, M_1, follows the tag on the back face of waiting module, M_2. The arrow from C_1 to T_2 shows the docking path to join M_1 and M_2. The waiting module hovers at a fixed position while the docking module tracks a tag and performs docking. The master module, $M^\star$, is the module in the assembled structure that will guide the structure in future dockings (here, $M^\star = M_2$).

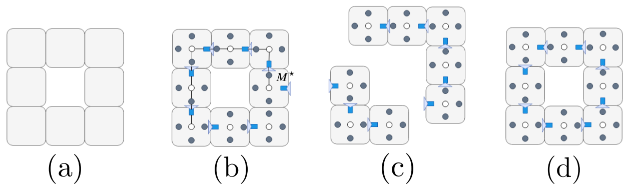

Fig. 3: (a) The desired structure. (b) Proper assembly of the desired structure, with a visual example of a path from one of the modules in the structures to the master represented by a black line. (c) A misalignment in camera position occurred in the smaller structure, preventing assembly. (d) No cameras point outward, rendering the structure unusable.

III. Vision-Based Self-Assembly

Challenge: A key challenge in achieving perception-based self-assembly is creating a plan that avoids formation deadlock; i.e., a sequence of docking actions that are infeasible for the modules to perform. Fig. 3 shows the desired structure (a), a feasible configuration of modules after assembly (b), and two infeasible assemblies (c,d). Grey squares represent modules, black circles represent tags, white circles represent the location of the modules, and blue squares represent the cameras. The two formation deadlocks depicted in Fig. 3c-d are: *i)* a formed structure has a shape such that no docking operations with other structures will result in the final desired structure; e.g., the docking structure would have to dock in the xz-plane and yz-plane simultaneously (Fig. 3c); or, *ii)* the formed structure has each module's camera pointing at a tag, leaving no cameras available for future dockings (Fig. 3d).

During a docking action, the docking module's camera has to be tracking one of the waiting module's tags. Options for assembly are limited because the docking module's camera can only see one of the non-camera sides of the waiting module. Further, any feasible assembly sequence must determine, for each assembly action, a) the direction each module will face in the assembled structure, and b) the individual substructures needed for each parallelized assembly step.

Approach: Our approach is based on two key insights. First, we observe that, although the docking module is always oriented towards the waiting module, its own orientation is invariant to the waiting module's orientation. Due to this invariance, we can iteratively determine the proper docking module orientation while disregarding waiting modules. This is accomplished alongside the computation of the sequence of docking actions. Second, by reinterpreting structures and sub-structures as *meta-modules*, we recursively split the structure into two parts: docking and waiting meta-modules. We orient the docking meta-module towards the waiting meta-module. In the following, we discuss our approach in more detail.

Definition 3. (**Meta-module**) *A meta-module is composed of a waiting ($\mathcal{M}_w$) and docking ($\mathcal{M}_d$) meta-module, along with the orientation information of the two contained meta-modules, where each contained meta-module represents a sub-structure of the whole structure. Each meta-module has a master module with an outward-facing camera.*

The docking actions needed to assemble a meta-module can be extracted by recursively splitting the component docking and waiting meta-modules, which each contain their own docking and waiting meta-modules. Thus, we recursively disassemble a meta-module to determine the docking action sequence needed to assemble a desired meta-module. In prior work [20], we recursively disassembled a structure by "chipping" away at its edges and stopped when a structure could no longer be disassembled (i.e., $|\mathcal{S}| = 1$). Generating the assembly plan is straightforward: we reverse the optimal disassembly plan. Robots act in a parallel to enable multiple docking actions in a single timestep. The efficiency of our algorithms depends on the number of *sequential* docking actions.

In the following subsections, we discuss our approach in detail. We begin with a strategy for finding the best master to guide the meta-module in future assembly while enforcing correct orientation (Section III-A). Then, we present a docking-step-optimized assembly algorithm that uses the meta-module concept to plan the assembly of a structure such that the number of sequential docking actions is minimized (Section III-B). Next, we adapt this technique to minimize the sum distance traveled by all robots by leveraging Hamiltonian paths (Section III-C). Finally, Section III-D presents an algorithm that uses the meta-module concept to find a full reconfiguration plan to transform a structure from one shape into another while also accounting for camera orientation.

A. Master Selection

First, we explain how to select a master module, $M^\star$, for a meta-module. That is, $M^\star$ is the module used to guide the meta-module in a future docking action.

To find an appropriate master for a structure, we find modules that are most "central" based on the Betweeness Centrality Algorithm [1], which produces a betweenness centrality degree for each node in a graph. Modules are convex and symmetric cuboids, thereby guaranteeing that mass is evenly distributed in a structure and that the betweenness centrality of modules correspond to how close they are to the structure's center of mass. Thus, by maximizing the betweenness centrality in the graph, we seek to find a set of modules that are closest to the center of the structure. For polygon-shaped agents (with n edges), modules of degree n are excluded as candidates because the master must have at least one face available for future dockings. Since our modules are cuboid, we exclude modules of degree four.

When multiple modules have the same betweenness centrality, we select the module with the fewest neighbors to be the master module; the least-connected module is likely to be at the boundary of the structure, where its field of view is free of obstructing modules and therefore available for future dockings. This minimax strategy is used to compute the master module as

$$M^\star = \operatorname*{argmin}_{M_i \in \mathcal{S}} \textsc{deg}(\operatorname*{argmax}_{M_i} \beta(M_i)),$$

where DEG is a function that returns the degree of each module and $\beta(M)$ is the output of the Betweeness Centrality algorithm. The optimal solution for $M^\star$ can be a set of modules of identical degree; in this case, we choose the median of the set.

Algorithm 1 Self-assembly algorithm to minimize sequential docking actions

```
Input: A geometric graph $\mathcal{G}$, meta module $\mathcal{M}$, and the master $M^\star$
Output: A docking action sequence $\mathcal{C}$, and orientation map $\mathcal{O}$
 1: $\mathcal{C} \leftarrow \{\}, \mathcal{O} \leftarrow \{(M^\star, \textsc{orientMaster}(M^\star))\}$
 2: TSEQUENCE($\mathcal{G}, \mathcal{M}, \mathcal{C}, \mathcal{O}$)
 3: return $\mathcal{C}, \mathcal{O}$
 4: function TSEQUENCE($\mathcal{G}, \mathcal{M}, \mathcal{C}, \mathcal{O}$)
 5:     if $|\mathcal{M}| = 1$ then
 6:         return
 7:     $\mathcal{P} \leftarrow \textsc{AllPossiblePartitions}(\mathcal{G}, \mathcal{M})$
 8:     $\mathcal{P}' \leftarrow \textsc{SortByBalance}(\mathcal{P})$
 9:     $(\mathcal{M}_d, \mathcal{M}_w) \leftarrow \mathcal{P}'[1]\ M^\star \in \mathcal{M}_w, M^\star \notin \mathcal{M}_d$
10:     $\mathcal{M}_n \leftarrow \textsc{NeighborModules}(\mathcal{G}, \mathcal{M}_d, \mathcal{M}_w)$
11:     $M^\star_{\mathsf{virt}} \leftarrow \textsc{MakeMaster}(\mathcal{G}, \mathcal{M}_n)$
12:     $\mathcal{O}[M^\star_{\mathsf{virt}}] \leftarrow \textsc{orient}(M^\star_{\mathsf{virt}}, \mathcal{M}_w)$
13:     TSEQUENCE($\mathcal{G}, \mathcal{M}_d, \mathcal{C}, \mathcal{O}$)
14:     TSEQUENCE($\mathcal{G}, \mathcal{M}_w, \mathcal{C}, \mathcal{O}$)
15:     $\mathcal{C}[\mathcal{M}] \leftarrow (breakline, \mathcal{M}_d, \mathcal{M}_w)$
16:     return
```

B. Minimizing Sequential Docking Actions

After selecting an appropriate master, we may compute an assembly sequence that minimizes the number of sequential docking actions. The approach is presented in Algorithm 1. First, we initialize the globally accessible docking action sequence $\mathcal{C}$ as an empty sequence, and the globally accessible orientation map $\mathcal{O}$ to contain only the master module $M^\star$ and its orientation; i.e., one of the directions in which the master would point away from its neighbor modules (Line 1). We then call the recursive function TSEQUENCE, which takes as arguments a geometric graph $\mathcal{G}$ representing the structure (including module positions), an initial meta-module representation $\mathcal{M}$, and the initial $\mathcal{C}$ and $\mathcal{O}$ from above (Line 2). In this function, we first check whether the meta-module consists of only one module - the base condition for our recursive function that is used by all algorithms we introduce in the remaining subsections (Line 5). If this base condition is not satisfied, at each timestep we generate all possible partitions of the graph and meta-module via ALLPOSSIBLEPARTITIONS, and then call SORTBYBALANCE to sort partitions in a balanced approach (as defined in Section III.B of [19]) (Lines 7–8). Then, we pick the best-balanced partition, and check which of its two meta-modules contains the master module; the one with the master module is the waiting meta-module while the other is the docking meta-module (Line 9). The docking meta-module is assigned a virtual master, $M^\star_{\mathsf{virt}}$, selected from the modules adjacent to the waiting meta-module (see Sec. III-A for details). Adjacent modules are the output of the NEIGHBORMODULES function, and the virtual master

Algorithm 2 Self-assembly algorithm to minimize total traveled distance

Input: A desired meta module $\mathcal{M}^+$, a geometric graph $\mathcal{G}$, meta module $\mathcal{M}$, a master $M^\star$, and the initial position graph $\mathcal{G}_0$
Output: A docking action sequence $\mathcal{C}$, and orientation map $\mathcal{O}$

```
1: $\mathcal{G} \leftarrow$ HAMILTONIANPATH($\mathcal{G}_0$)
2: $\mathcal{C} \leftarrow \{\}, \mathcal{O} \leftarrow \{(M^\star, \text{ORIENTMASTER}(M^\star))\}$
3: DSEQUENCE($\mathcal{M}^+, \mathcal{G}, \mathcal{M}, \mathcal{C}, \mathcal{O}$)
4: return $\mathcal{C}, \mathcal{O}$
5: function DSEQUENCE($\mathcal{M}^+, \mathcal{G}, \mathcal{M}, \mathcal{C}, \mathcal{O}$)
6:     if $|\mathcal{M}| = 1$ then
7:         return
8:     $\mathcal{P} \leftarrow$ ALLPOSSIBLEPATHPARTITIONS($\mathcal{G}, \mathcal{M}$)
9:     $\mathcal{P}^+ \leftarrow$ ALLPOSSIBLEPARTITIONS($\mathcal{M}^+, \mathcal{M}$)
10:    $\mathcal{P}' \leftarrow$ SORTBYWEIGHT($\mathcal{P}$)
11:    for $(\mathcal{M}_d, \mathcal{M}_w) \in \mathcal{P}'$ do $M^\star \in \mathcal{M}_w, M^\star \notin \mathcal{M}_d$
12:        if not VALIDDIVISION($\mathcal{M}_d, \mathcal{M}_w, \mathcal{P}^+$) then
13:            continue
14:        $M^\star_{\text{virt}} \leftarrow$ TREENEIGHBORMODULE($\mathcal{G}, \mathcal{M}_d, \mathcal{M}_w$)
15:        $\mathcal{O}[M^\star_{\text{virt}}] \leftarrow$ ORIENT($M^\star_{\text{virt}}, \mathcal{M}^+_w$)
16:        DSEQUENCE($\mathcal{M}^+_d, \mathcal{G}, \mathcal{M}_d, \mathcal{C}, \mathcal{O}$)
17:        DSEQUENCE($\mathcal{M}^+_w, \mathcal{G}, \mathcal{M}_w, \mathcal{C}, \mathcal{O}$)
18:        $\mathcal{C}[(\mathcal{M}^+, \mathcal{M})] \leftarrow (breakline, \mathcal{M}_d, \mathcal{M}_w)$
19:        return
20: function VALIDDIVISION($\mathcal{M}_d, \mathcal{M}_w, \mathcal{P}^+$)
21:    for $(\mathcal{M}^+_d, \mathcal{M}^+_w) \in \mathcal{P}^+$ do
22:        if $|\mathcal{M}^+_w| = |\mathcal{M}_w|$ & $|\mathcal{M}^+_d| = |\mathcal{M}_d|$ & $(M^\star \in \mathcal{M}^+_w)$ then
23:            return true
24:    return false
```

Algorithm 3 Self-reconfiguration algorithm

Input: A meta module $\mathcal{M}$, a desired meta module $\mathcal{M}^+$
Output: Assembly and disassembly action sequences $\mathcal{C}_a$ and $\mathcal{C}_d$

```
1: $\mathcal{C}_a \leftarrow \{\}, \mathcal{C}_d \leftarrow \{\}$
2: RECONFIGURE($\mathcal{M}, \mathcal{M}^+, \mathcal{C}_a, \mathcal{C}_d$)
3: return $\mathcal{C}_a, \mathcal{C}_d$
4: function RECONFIGURE($\mathcal{M}, \mathcal{M}^+, \mathcal{C}_a, \mathcal{C}_d$)
5:     if $|\mathcal{M}| = 1$ then
6:         return
7:     $(\mathcal{M}_d, \mathcal{M}_w) \leftarrow \mathcal{M}$
8:     CHECKPAIR($\mathcal{M}_d, \mathcal{M}_w$)
9: function CHECKPAIR($\mathcal{M}_d, \mathcal{M}_w$)
10:    $\mathcal{T} \leftarrow \{\mathcal{M}_d, \mathcal{M}_w\}$
11:    $\mathcal{T}_R \leftarrow$ RECONFIGURABLE($\mathcal{T}, \mathcal{M}^+$)
12:    if $|\mathcal{T}_R| = 2$ then
13:        $\mathcal{C}_a[\mathcal{M}] \leftarrow (\mathcal{M}_d, \mathcal{M}_w)$
14:    else
15:        $\mathcal{C}_d[(\mathcal{M}_d, \mathcal{M}_w)] \leftarrow \mathcal{M}$
16:        if $|\mathcal{T}_R| \neq \varnothing$ then
17:            $\mathcal{C}_a[\mathcal{M}] \leftarrow \mathcal{M}_i, \quad \mathcal{M}_i \in \mathcal{T}_R$
18:        for $\mathcal{M}_i \notin \mathcal{T}_R$ do
19:            RECONFIGURE($\mathcal{M}_i, \mathcal{M}^+, \mathcal{C}_a, \mathcal{C}_d$)
20: function RECONFIGURABLE($\mathcal{T}, \mathcal{M}^+$)
21:    $\mathcal{T}_R \leftarrow \varnothing$
22:    for $\mathcal{M}_i \in \mathcal{T}$ do
23:        if all modules in $\mathcal{M}_i$ can be placed in correct spot in $\mathcal{M}^+$ and master
           of $\mathcal{M}_i$ is correctly oriented then
24:            $\mathcal{T}_R \leftarrow \mathcal{T}_R \cup \mathcal{M}_i$
25:    return $\mathcal{T}_R$
```

is assigned in the MAKEMASTER function (Lines 10–11). The ORIENT function orients the virtual master towards the waiting meta module (Line 12).

Next, recursive TSEQUENCE calls are made on the waiting and docking meta-modules such that they properly orient their own modules and calculate the next iteration to be added to the action sequence (Lines 13–14). After all lower-level recursive TSEQUENCE calls return, the action sequence at the current iteration is assigned the proper docking and waiting meta-modules and breakline across which they would both assemble (Line 15).

Analysis. Algorithm 1 has bounded recursive depth of $\Theta(\log n)$, where n is the number of modules. This is because modules move to the expanded configuration and are evenly spaced before undertaking assembly, so successive assembly operations can be performed by meta-modules that are "neighbors". This allows optimal parallelization: for every assembly, meta-module size changes by a factor of two.

C. Distance-Minimal Sequencing

In minimizing the number of sequential assembly steps, modules move to the expanded configuration of the desired configuration before starting assembly. In certain tasks, however, moving to the expanded configuration of a structure is undesirable. For instance, it makes more sense to minimize total distance traveled when distances between modules are enormous (e.g., across countries). Here, since modules no longer move to the expanded configuration of the desired structure, we cannot use the same meta-module splitting approach as in Algorithm 1.

Forming a distance-minimal geometric representation that subscribes to the meta-module principle requires finding the minimum Hamiltonian path in the graph - an NP-complete problem. We use an algorithm [8] that adapts Christofides' Hamiltonian circuit finding algorithm [2] to finding Hamiltonian paths. We can divide the shortest Hamiltonian path into sub-groups of modules of arbitrary size, leveraging knowledge of the initial positions of modules. Such division allows us to obtain groups of modules such that the portion of the shortest Hamiltonian path (i.e., the fraction of total distance in the path) in each group is roughly equal, meaning that the partitions are balanced with respect to distance. Moreover, this approach of recovering the geometry can be extended to robots that are constructed differently (i.e. robots of different convex polygonal shapes). For instance, we could substitute the shortest Hamiltonian path with a degree-constrained minimum spanning tree if the modules have more degrees of freedom. Once we obtain the graph representing the Hamiltonian path, we use the graph in a similar manner as in time-minimal sequencing to compute the displacement-minimal sequencing.

Algorithm 2 presents the algorithm for distance-minimal sequencing using the above approach. The algorithm starts by initializing the docking action sequence map $\mathcal{C}$ and the orientation map $\mathcal{O}$ (Line 2). The desired meta-module $\mathcal{M}^+$, geometric graph $\mathcal{G}$, current meta-module $\mathcal{M}$, together with the initialized output maps, are passed into DSEQUENCE (Line 3). The base case (meta-module of size one) is handled in the same manner as Algorithm 1. Lines 8-10 are similar to the corresponding ones in time-based assembly sequencing. However, at each recursive layer, instead of partitioning the structure by balancing the number of modules they contain, we partition the graph (describing the minimum path) by balancing the sum of distances between nodes in partitions.

We iterate over the partitions and use VALIDDIVISION to check which ones have the necessary number of modules to fit into the desired structure partitions and whether the master module is in the waiting sub-structure. If the conditions are not satisfied, we move to the next-best partition. Otherwise, the two resulting substructures are defined as meta-modules, and the node in the docking meta module that had an edge to the waiting meta-module severed during path partitioning becomes the virtual master. In ORIENT, the virtual master is oriented towards the waiting meta-module (perpendicular to the breakline that cuts the desired structure).

The procedure recurses over the meta-modules in Lines 16-17 and stores the returned assembly actions and orientations into the action sequence $\mathcal{C}$ and orientation map $\mathcal{O}$.

Analysis. Due to their similar structures, the best-case recursive depth produced by Algorithm 2 is the same as that of the time-minimal sequencing algorithm, which is $\Omega(\log n)$, where n is the number of modules. However, in the worst case, recursive depth is $O(n)$. This occurs when only one module can conduct assembly in each time step. This could happen when the modules are arranged in a dispersed line such that the distance between every additional pair of modules is exponentially increasing, and the structure being assembled is a line. In this scenario, the most balanced partition in the structure partitions would contain a waiting structure that has all the modules in the original structure except for one module that is too far away. Thus, for every timestep, only one module conducts assembly.

D. Reconfiguration

Our reconfiguration algorithm tries to disassemble an initial structure as little as possible during the reconfiguration process. That is, it successively disassembles meta-modules; for each meta-module, it is only disassembled further if it cannot be "placed" in the correct position in the new desired structure as is. By the word "placed", we mean that every physical module of the meta-module can be placed into its new desired position, no formation deadlock is caused by the placement, and the docking module can be oriented towards the waiting module.

This high-level idea has been considered in our prior work [5], but instead of using a global positioning system like Vicon as [5] does, this work leverages agent-local vision for high-accuracy maneuvering. Since the docking module is invariant to the inner modules' orientations (barring the single invalid case when the waiting module's camera is facing the docking module's camera), we only need to account for the orientation of the master module within the meta-module when placing the modules. The complete procedure for reconfiguration is shown in Algorithm 3.

As Algorithm 3 shows, instead of using a single docking action sequence, we use two (initially empty) action sequences—an assembly sequence $\mathcal{C}_a$ and a disassembly sequence $\mathcal{C}_d$—to describe the complete reconfiguration process. The reconfiguration procedure RECONFIGURE immediately returns if the base condition (i.e., the meta-module consists of only one module) is satisfied (Lines 5-6). Otherwise, we split the current meta-module representation into a pair of docking and waiting meta-modules ($\mathcal{M}_d$ and $\mathcal{M}_w$), and then call the CHECKPAIR function to check for a possible fitting within the desired meta-module (Lines 7–8).

In CHECKPAIR, $\mathcal{T}_R$ stores the set of meta-modules that can fit in the desired meta-module $\mathcal{M}^+$. It is assigned to the set of valid reconfigurable meta-modules computed by RECONFIGURABLE (Line 11).

The RECONFIGURABLE subprocedure initially assigns $\mathcal{T}_R$ to be an empty set (Line 21). It then iterates over the docking and waiting meta-modules to determine whether they will correctly fit into the desired meta-module $\mathcal{M}^+$ (Line 22). Here, the term "correctly fit" refers to when all modules can be placed in the desired positions and the master is correctly oriented. For the docking meta-module, this means the master can be properly oriented relative to the waiting meta-module. For the waiting meta-module, this means the master's field of view is unobstructed and that it does not border the docking meta-module. The meta-modules that satisfy these conditions are placed in $\mathcal{T}_R$ (Line 24).

Once $\mathcal{T}_R$ is computed by RECONFIGURABLE, we check whether the docking and waiting meta-modules contained within $\mathcal{T}_R$ can correctly fit into the desired meta-module (Line 12). If so, we add them both to the assembly action sequence $\mathcal{C}_a$ (Line 13) and return from the CHECKPAIR function. Otherwise, we add them to the disassembly action sequence $\mathcal{C}_d$ (Line 14). If either of the meta-modules fits, then we add it to the assembly action sequence (Line 17). Those modules that could not be correctly reconfigured at the current recursive layer are further split until they do correctly fit. The algorithm will break down a meta-module only as much as is needed—if a meta-module can fit as is in the desired meta-module, it will not be further divided.

Analysis. Our algorithm has a recursive depth of $\Theta(\log n)$ in both best and worst cases. Reconfiguration uses docking-step-minimal sequencing (Algorithm 1) as its backbone. Thus, we know that for both disassembly and assembly, modules will move into an expanded configuration that allows us to optimally configure partitions such that for each recursive layer, partition size changes by a factor of two. There is a constant multiplying factor of two (i.e., $\Theta(2\log n)$) since we perform both disassembly and assembly, but this does not affect the asymptotic recursive depth.

IV. EXPERIMENTAL VALIDATION

To evaluate our algorithms, we performed a series of experiments in simulation. Our evaluation focuses on two key questions: 1) How do the algorithms perform on a range of structure sizes and shapes? and 2) How well do they work when more computationally intense, but also more realistic, physics is accounted for? We had to split the evaluation into two parts because of the computational power required for more realistic simulation. Thus, we implemented two simulators: a 2-D simulator in Python to evaluate large and diverse structures; and a more computationally intensive Gazebo 3-D simulator to evaluate the algorithms in a more realistic environment. Code for both simulations is available at `https://github.com/swarmslab/modquad-vi-sims/`.

Setup. All experiments ran on a machine equipped with an Intel i7-8650 CPU and 16 GB of RAM. Our Python simulator was built using the `networkx` library. The more

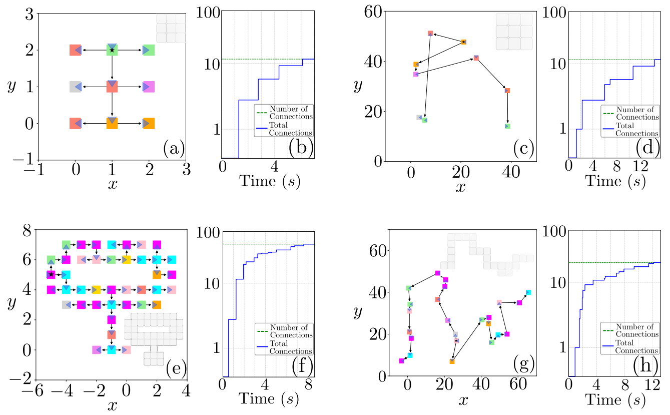

Fig. 4: (a,c,e,g) present planned assembly sequences (arrows show assembly direction). The master was hand-picked in (e) - the hole in the structure affected the selection process. (b,d,f,h) show connections made as time increases for the structures of (a,c,e,g), respectively.

realistic Gazebo simulator was built on MAVROS, itself a ROS [17] package. In Gazebo, each quadrotor was equipped with virtual tags, a camera, and a protective cage; we also simulated the quadrotor hardware. We also placed 4 magnets on each vertical face of a module and gave each the same magnetic moment. When a docking module gets close to the waiting module's WhyCon tag, a "snap" action occurs; i.e., the magnetic force between the two modules increases exponentially, indicating the completion of docking.

Results. We first discuss results from our Python simulations. We tested three types of configurations: a 9-module square landing platform, a 41-module hourglass-shaped structure with a hole in the middle, and a 24-module bridge network. We refer to results of Algorithm 1 as *docking-step-minimal* and Algorithm 2 as *distance-minimal*. Figs. 4a, 4c, 4e, and 4g show the assembly trees with the randomly colored modules with the target structure shown in grey. The modules' docking directions are shown as arrows. The orientation of the blue triangles (representing the camera view) shows the docking direction.

For the landing platform, we present both docking-step-minimal (Fig. 4a) and distance-minimal (Fig. 4c) assemblies. Both assemblies require a series of four assembly "docking steps", where each "step" contains multiple actual docking operations. We can verify the number of steps is optimal by qualitatively comparing the theoretically optimal number of steps needed to the actual number of steps taken to perform the assembly: $\log_2(9) = 3.2$, where 9 is number of modules in the structure; since step counts must be whole numbers, the optimal number of steps is $\lceil 3.2 \rceil = 4$.

Fig. 6a shows the optimal Hamiltonian path for the same structure as in Fig. 4c. Note that the two assembly sequences visually resemble one another. In the worst case, Christofides' path generation is $\frac{5}{3}$ optimal [8] (i.e., the generated path's length will always be within a factor of $\frac{5}{3} \approx 1.67\times$ the optimal solution's length). For this particular structure, the solution was $1.37\times$ optimal displacement. Next, we examine more complex structures. Fig. 4e shows the assembly sequence of the 41-module hourglass structure as determined by Algorithm 1. It took seven docking steps to assemble all 41 modules into the desired structure. $\lceil \log_2(41) = 6 \rceil$ is the expected optimal number of steps, but due to the odd number of modules we have one "left over" module that adds a step.

Fig. 4g shows the 24-module bridge assembly sequence, as determined by Algorithm 2. These modules were assembled in seven docking steps (a near-optimal number of steps, $\lceil \log_2(24) \rceil = 5$, despite distance minimization not prioritizing docking-step minimization). Again, by visually comparing to the optimal path for this structure presented in Fig. 6b, we observe that the heuristics-produced result is similar to the optimal one. In more quantitative terms, the heuristics-based solution in Fig. 4g was $1.043\times$ optimal, well below the Christofides' guaranteed bound of $1.67\times$ optimal. Thus, our technique works and can scale well to a large number of modules using both the docking-step- and distance-minimal techniques.

Next, we examine trends during assembly. Figs. 4b, 4d, 4f, and 4h present the number of established magnetic connections as a function of time, where a "connection" is defined as a pair of modules that are magnetically linked. As expected, due to parallelization, we see large jumps in the number of connections early on. As meta-modules become larger, the number of additional connections established per unit time declines, ultimately approaching the number of connections needed to form the desired structure (i.e., the dashed green line).

To evaluate our reconfiguration algorithm, we present Fig. 5. Fig. 5a shows the initial square structure, Fig. 5b shows the structure in its *maximally disassembled state*, where many of the modules remain a part of a larger sub-structure because the entire sub-structure can be placed in the new full desired structure. Fig. 5c shows the final structure reconfigured to. The full reconfiguration process took 8 parallelized steps (4 steps each for disassembly and assembly) instead of 9 steps for complete disassembly and assembly, showing how the algorithm breaks down the structure only as much as it needs to for the reconfiguration process. For even larger structure reconfigurations, the number of steps saved are expected to be more substantial.

Finally, we look at our Gazebo simulations. Fig. 1 shows the result structure of a time-minimal sequencing experiment with five modules. The structure completed assembly using the onboard camera and IMU of each quadrotor module. The success of the assembly process in the more physically accurate Gazebo simulator supports the algorithms' potential to be applied in the real world. Both Gazebo and Python simulation recordings are available at: `https://youtu.be/ayhsNq9YlPo`.

Summary. Our experiments have supported the theoretical efficiencies we were expecting. Recursive depth was optimal or near-optimal for both docking-step- and distance-minimal assembly. In distance-minimal assembly, we confirmed that our heuristics-based approach stays within the $1.67\times$ optimal bound that Christofides' algorithm guarantees. Finally, for reconfiguration, our experiments show efficiency improvement

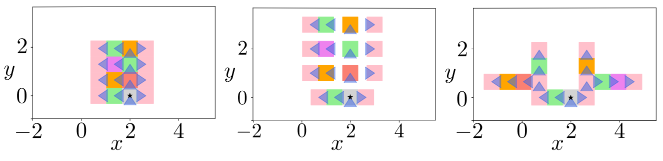

Fig. 5: Reconfiguring a square structure into a double horseshoe structure. (a) Initial lattice structure; (b) The disassembled structure; and (c) reconfigured structure.

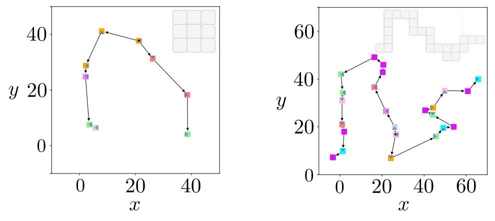

Fig. 6: Optimal geometrically recovered Hamiltonian paths for Figures 4c and 4g (target structures shown in grey).

over the "complete" assembly adopted in previous work.

There are some important differences between the techniques; e.g., the scenarios under which docking-step and distance-minimal techniques work best differ. In docking-step-minimal assembly, closely packed modules can easily parallelize assembly, but modules close to one another may not always dock to one another, thereby making the assembly *not* distance-optimal. However, densely-packed modules often do not have a large distance to travel, so in such cases the additional displacement caused by the docking-step-minimal technique may be negligible.

Additional steps in assembly become increasingly expensive as distances between modules increases as other resource constraints (e.g., fuel) come to bear. Here, it is prudent to allow for larger recursive depth but lower consumption of other resources. Recall, though, that the distance-minimal technique is still highly parallelized, so we often get good time-step performance. Fig. 4a and 4c had identical time-step-performance, and even for the complex bridge network of Fig. 4g, we only require two additional time steps relative to optimal.

V. Conclusion and Future work

In this paper, we present three algorithms for self-assembly and self-reconfiguration that combine global and local positioning systems. The algorithms use the lower-accuracy global positioning system to determine the sequence of steps, and an on-board relative positioning system for precise maneuvering and docking actions. The algorithms take into account the constraints of the local sensors and use heuristics to increase efficiency in solving the combinatorial problem. We presented algorithms for docking-step-minimal self-assembly, distance-minimal self-assembly, and the self-reconfiguration of a modular structure. Our simulations in 2-D and 3-D show that the algorithms scale with the number of modules and structure shape. In future work, we plan to explore approaches that can be performed in cluttered environments and executed in a distributed fashion.

References

[1] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2):136–145, 2008.
[2] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, CMU, 1976.
[3] M. J. Duffy and T. C. Samaritano. The lift! project–modular, electric vertical lift system with ground power tether. In *33rd AIAA Applied Aerodynamics Conference*, 2015.
[4] B. Gabrich, D. Saldaa, V. Kumar, and M. Yim. A flying gripper based on cuboid modular robots. In *ICRA 2018*.
[5] N. Gandhi, D. Saldaña, V. Kumar, and L. T. X. Phan. Self-reconfiguration in response to faults in modular aerial systems. *IEEE Robotics and Automation Letters*, 5:2522–2529, 2020.
[6] B. Gheneti, S. Park, R. Kelly, D. Meyers, P. Leoni, C. Ratti, and D. Rus. Trajectory planning for the shapeshifting of autonomous surface vessels. In *MRS 2019*, pages 76–82.
[7] L. Grimstad and P. From. The Thorvald II agricultural robotic system. *Robotics*, 6:24, 2017.
[8] J. A. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Oper. Res. Lett.*, 10(5):291295, July 1991.
[9] G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit. An end-to-end system for accomplishing tasks with modular robots: Perspectives for the AI community. IJCAI'17.
[10] B. Li, S. Ma, J. Liu, M. Wang, T. Liu, and Y. Wang. AMOEBA-I: A shape-shifting modular robot for urban search and rescue. *Advanced Robotics*, 23:1057 – 1083, 2009.
[11] G. Li, B. Gabrich, D. Saldaña, J. Das, V. Kumar, and M. Yim. ModQuad-Vi: A vision-based self-assembling modular quadrotor. In *ICRA 2019*.
[12] L. A. Mateos, W. Wang, B. Gheneti, F. Duarte, C. Ratti, and D. Rus. Autonomous latching system for robotic boats. In *ICRA 2019*.
[13] D. Mellinger, M. Shomin, N. Michael, and V. Kumar. Cooperative grasping and transport using multiple quadrotors. In *DARS 2010*.
[14] M. Nitsche, T. Krajník, P. Čížek, M. Mejail, and T. Duckett. Whycon: An efficent, marker-based localization system. In *IROS Workshop on Open Source Aerial Robotics*, 2015.
[15] I. O'Hara, J. Paulos, J. Davey, N. Eckenstein, N. Doshi, T. Tosun, J. Greco, J. Seo, M. Turpin, V. Kumar, and M. Yim. Self-assembly of a swarm of autonomous boats into floating structures. In *ICRA 2014*.
[16] R. Oung, F. Bourgault, M. Donovan, and R. D'Andrea. The distributed flight array. In *ICRA 2010*.
[17] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
[18] D. Saldaña, B. Gabrich, G. Li, M. Yim, and V. Kumar. ModQuad: The flying modular structure that self-assembles in midair. In *ICRA 2018*.
[19] D. Saldaña, B. Gabrich, M. Whitzer, A. Prorok, M. F. M. Campos, M. Yim, and V. Kumar. A decentralized algorithm for assembling structures with modular robots. In *IROS 2017*.
[20] D. Saldaña, P. Gupta, and V. Kumar. Design and control of aerial modules for inflight self-disassembly. *IEEE Robotics and Automation Letters*, 4(4):3410–3417, 2019.
[21] J. Thomas, G. Loianno, K. Daniilidis, and V. Kumar. Visual servoing of quadrotors for perching by hanging from cylindrical objects. *IEEE Robotics and Automation Letters*, 1(1):57–64, 2016.
[22] J. Thomas, M. Pope, G. Loianno, E. Hawkes, M. Estrada, H. Jiang, M. Cutkosky, and V. Kumar. Aggressive flight for perching on inclined surfaces. *Journal of Mechanisms and Robotics*, 8, 12 2015.
[23] J. Werfel and R. Nagpal. Three-dimensional construction with mobile robots and modular blocks. *I. J. Robotics Res.*, 27(3-4):463–479, 2008.
[24] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robot. Automat. Mag.*, 14(1):43–52, 2007.