A Decentralized Algorithm for Assembling Structures with Modular Robots

David Saldaña[1], Bruno Gabrich[1], Michael Whitzer[1], Amanda Prorok[1], Mario F. M. Campos[2], Mark Yim[1], and Vijay Kumar[1].

***Abstract*— Recent work in the field of bio-inspired robotic systems has introduced designs for modular robots that are able to assemble into structures (e.g., bridges, landing platforms, fences) using their bodies as the building components. Yet, it remains an open question as to how to program large swarms of robotic modules so that the assembly task is performed as efficiently as possible. Moreover, the problem of designing assembly algorithms is compounded by the scale of these systems, and by the lack of centralized guidance in unstructured environments. The main contribution of this work is a *decentralized* algorithm to assemble structures with modular robots. Importantly, we coordinate the robots so that docking actions can be parallelized. We show the correctness of our algorithm, and we demonstrate its scalability and generality through multiple scenarios in simulation. Experiments on physical robots demonstrate the validity of our approach in real-world settings.**

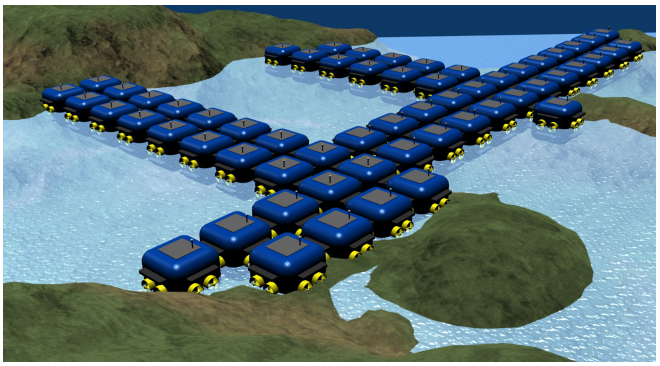


Fig. 1. A large team of modular robots that connects four different islands by assembling a bridge.

I. Introduction

In nature, we see how a large number of ants are capable of constructing structures using their bodies as the building components. This capability allows them to rapidly build temporary bridges to connect disjoint areas in order to transport food and resources to their colonies. One of the challenges in modular robotics is to develop mechanisms and algorithms for autonomous robots that imitate these types of behaviors found in nature. Specific formations and attaching mechanisms allow multiple robots to build structures on land or over water. Depending on the underlying real-world problem, the utility of these structures will vary — for example, bridges will connect disjoint lands, floating surfaces at sea serve as landing platforms, and walls serve as physical delimitations. Many modular systems, such as those presented in [1], [2], [3], have developed modular robots that are capable of moving in space and assembling. Docking all robots at the same time is a difficult task, since undesired attachments can be generated due to alignment problems and inaccurate motion [4], [5]. For this reason, a conservative approach, to avoid collisions and undesired attachments, satisfies that *(i)* only pairs of attached groups are able to dock at a time, and *(ii)* robots have to slow down in order to align and orient themselves in the proper direction to guarantee reliable docking. In order to satisfy these conditions, current approaches [6] consider the addition of robots one-by-one to form a modular robotic chain. However, this process is inefficient in terms of construction speed due to the fact that the structure's formation time increases linearly with the number of required robots.

Let us consider the modular bridge in Figure 1, where a team of sixty modular floating boats creates a bridge to connect three disjointed islands. In this environment, a single robot is able to easily move around, but it is not capable of connecting any pair of islands on its own. The collective effort enables the swarm to assemble different structural configurations in a fast manner. Both the maneuverability of single agents and the method used for the assembly of multi-robot structures greatly effects the assembly efficiency. In this paper, we focus on the development of an algorithm that can be deployed on a large number of modular robots, with the goal of assembling structures using the robot bodies as the building units.

Our approach can be applied to any aquatic or ground modular robot while satisfying the conditions of being square and holonomic. However, this holonomic condition can be relaxed by allowing rotations of the modular structure, such as the modular robot presented in [4].

A. Related work

In the robotics literature, multi-robot and swarm robotic systems have been widely studied [7], [8], [9]. In these

[1] D. Saldaña, B. Gabrich, M. Whitzer, A. Prorok, M. Yim and V. Kumar are with the GRASP Laboratory, University of Pennsylvania, Philadelphia, PA, USA: {dsaldana, brunot, mwhitzer, prorok, yim, kumar}@seas.upenn.edu

[2]M.F.M. Campos is with the VeRLab Laboratory, Universidade Federal de Minas Gerais, MG, Brazil: {saldana, mario}@dcc.ufmg.br

*The authors gratefully acknowledge the support of the Colombian agency COLCIENCIAS, and the Brazilian agencies: CAPES, CNPq and FAPEMIG. We also acknowledge the support of DARPA grant HR00111520020, ONR grants N00014-15-1-2115 and N00014-14-1-0510, ARL grant W911NF-08-2-0004, NSF grant IIS-1426840, and TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

works, distributed controllers are proposed to create formations based on the number of robots, but they do not consider docking. The concept of attachment in swarm systems was introduced in [10], where multiple robots join forces to extend their capabilities. Some work focuses on building structures based on modular robots; Cucu et al. [11] present a bio-inspired ground robot design that builds pyramid or chain structures. In this design, the robots can climb, move, and attach among themselves in order to progressively build stable structures. In [12] a cubic module is presented that can form various arrangements, but it is difficult to move and accurately localize. In [13], a flying array system is proposed where the robot design is based on an omni-directional ground platform with a propeller. The agents are manually docked together on the ground in order to build a flying array to increase the payload capabilities of the robot team, and to reduce the number of failure points. However, this work primarily focused on how to enable the modules to fly as a cohesive unit, and not on how to efficiently join the modules. In other work, multiple robots can build structures based on attachable units, like bricks, that they stack together [14], [15].

Our main contribution is a new decentralized algorithm to assemble structures with multiple modular robots. In contrast to prior work, our method is focused on guaranteeing that the desired structure is assembled by parallelizing docking actions while avoiding collisions and undesired attachments.

II. Problem Statement

We have a team of n modular robots in the Euclidean space $\mathbb{R}^2$. The location of each robot i is denoted by $x_i \in \mathbb{R}^2$, and its orientation is denoted by $\theta_i \in [0, 2\pi]$, $i = 1, ..., n$. The module has a square shape with a width equal to w. We assume that the robots are holonomic and their control input follows first order dynamics

$$\dot{x}_i = u_i.$$

The attachment mechanism works as a passive actuator that is triggered when two or more modular robots are in contact (e.g. an attaching mechanism based on magnets). Since the robots have a square shape, they have four edges on which horizontal and vertical attachments can be made. Each robot is equipped with localization and attaching sensors to allow them to know their location in planar space and to identify if it is attached to other robots. The attaching sensor can be a physical contact sensor or the combination of local communication and location information. For instance, robot i knows if it is attached to another robot j when $\|x_i - x_j\| = w$, where $\|\cdot\|$ denotes the Euclidean norm. Additionally, each robot is also equipped with a relative location sensor that accurately identifies nearby robot locations. This sensor is mainly used during alignment for the docking action. We refer to *docking action* as the motion behavior that one or multiple joined robots perform to attach to another non-empty set of joined robots. During this action, motion and maneuverability are required to precisely align the attachment mechanisms. For this reason, the docking action is the most time-consuming task.

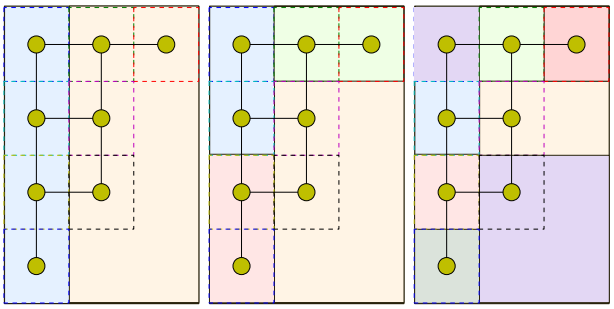


Fig. 2. Decomposing the bounding box of a *feasible structure* into sub-rectangles. The disks represent the target points $\mathcal{Z}$ and their attachments are represented by the continuous black lines. The bounding box surrounds the structure and its sub-rectangles are highlighted by the background colors. Panel (a) shows the decomposition of the bounding box into two sub-rectangles. Panel (b) shows the decomposition of the initial sub-rectangles, and Panel (c) shows the final decomposition, where each sub-structure only contains a single module.

Assumption 1. *Robots can move rapidly within their workspace. In contrast, docking actions must be executed with precision. As a consequence, we assume that the time robots spend moving throughout space is negligible.*

The desired structure is specified by a set of target points $\mathcal{Z} = \{z_1, ..., z_n\}$, $z_i \in \mathbb{R}^2$. Since these target points define the final locations of the modules, they should satisfy the following assumption.

Assumption 2. *The set of target points $\mathcal{Z}$, $|\mathcal{Z}| > 0$ satisfies:*

- *The distance between two adjacent points is w.*
- *The target points form a single connected structure.*
- *Let θ be the* structure orientation*, for any pair of points $z_i, z_j \in \mathcal{Z}$, s.t. $\|z_i - z_j\| = w$, there exists a straight line with angle θ or $\theta + \pi/2$ that crosses both points. This means that the points are aligned in such a way that only horizontal or vertical connections are allowed.*

As it was aforementioned, we want to assemble structures by only docking pairs of independent modules. The types of structures that can be assembled is constrained because we are only using horizontal or vertical docking actions. In this paper, a *feasible structure* defines a compound structure whose bounding box can be decomposed into a pair of sub-rectangles, where each substructure is also a connected sub-structure. Recursively, each sub-rectangle is also a compound structure, meaning that its sub-rectangles are also connected components. This property must be maintained until each sub-structure is only composed by a single robot. Figure 2 exemplifies a compound structure and its sub-rectangles. In Figure 2(a), we can see the bounding box and its two sub-rectangles, where each sub-structure is also decomposed, as illustrated in Figures 2(b) and 2(c).

The robots start at arbitrary locations as isolated units. We assume that any pair of robot locations, (x_i, x_j), satisfies the minimum separation distance $\|x_i - x_j\| > 2w$. Using these conditions, the problem addressed in this paper will now be presented.

Problem 1. *Given a desired feasible structure, specified by a set of target points $\mathcal{Z}$, find the collision free motions and assembly sequence of $|\mathcal{Z}|$ modular robots in order to assemble the given structure.*

We present an efficient control algorithm to solve Problem 1 in the next section.

III. Parallel Assembly Algorithm

We propose a *Parallel Assembly Algorithm (PAA)* that controls a team of isolated modular robots to construct a desired structure while avoiding collisions and unexpected attachments. Since the docking actions are the time-consuming part of the assembly process, our method focuses on parallelizing these actions. In order to assemble a desired structure, the robots move from their initial arbitrary locations to specific locations in a formation. Following this, the robots then apply a sequence of docking actions to finally create a single connected component.

For convenience and without loss of generality, we state all our following algorithms assuming that each robot's orientation is zero, $\theta_i = 0$. There is no loss of generality in this assumption because any desired structure angle θ can be oriented to zero by a simple axial rotation. Suppose $\theta \neq 0$, and consider the change of variables $x'(t) = R_\theta x(t)$, $\theta_i = 0$, where R_θ is a rotation matrix. Therefore, generality is not lost and we can assume that $\theta = 0$ and $\theta_i = 0$.

In the beginning, the robots start in arbitrary locations and all of them receive the desired structure as a set of target points $\mathcal{Z}$ (they do not know where they are allocated yet). Then, they follow our decentralized method which is based on the following three stages.

A. Stage 1: Navigating to an Expanded Configuration

Attaching all robots at the same time is a difficult task due to alignment problems and inaccurate movements. For this reason, we first move the robots from their initial locations to an expanded configuration of the desired target locations.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph that represents the structure, where the vertices $\mathcal{V} = \{1, ..., n\}$ are associated to the target points of the robots, and the edges symbolize the desired physical connections between each pair of robots and are represented as $\mathcal{E} = \{(i,j) | \|z_i - z_j\| = w, \forall i,j \in \mathcal{V}\}$. The geodesic distance between a pair of nodes refers to the number of edges in the shortest path. Let $c \in \mathcal{V}$ be a vertex with minimum eccentricity, i.e., a vertex in the center of the graph which has the minimum geodesic distance to reach any other vertex in $\mathcal{V}$ [16]. The associated reference point to the node c is denoted by z_c. Therefore, we compute the expanded set of target points as

$$\mathcal{Z}^E = \{2(z - z_c) + z_c, \forall z \in \mathcal{Z}\}. \quad (1)$$

In this new set of target points, each pair of neighboring points increases their distance from w to $2w$.

The robots are then moved from their arbitrary locations to the expanded target locations $\mathcal{Z}^E$. We use the Decentralized Concurrent Assignment and Planning of Trajectories (D-CAPT) algorithm [17] as a decentralized method to allocate and move robots from their initial locations to the expanded target points $\mathcal{Z}^E$ by minimizing the maximum traveling distance and avoiding collisions. By using D-CAPT, we ensure that all the target points will be properly allocated and the modules will not collide while navigating to the extended target points $\mathcal{Z}^E$. This expanded configuration allows the robots to identity the place where each of them belongs in the final configuration.

B. Stage 2: Computing the Assembly Tree

Since the main bottleneck during the assembly process is the docking actions, we want to compute an efficient and feasible sequence of these actions. Our approach is focused on parallelizing docking actions that can be performed independently. In other words, we want to start docking pairs of single robots, and in the next step dock pairs of sets that contain two already attached robots, then pairs of four, and continuing in this way until the whole team is attached.

The *assembly tree* determines the sequence of docking actions that each subset of robots must follow. It can be computed in a decentralized manner such that each robot i uses its current location in the expanded graph $x_i \in \mathcal{Z}^E$ to compute its own branch in the assembly tree.

We call a *component*, denoted by $\mathcal{C}$, a set of attached robots that behave as a single rigid body. A *partition* traces a vertical or a horizontal line which divides a component into two disjoint sets $\mathcal{C}_1 \cup \mathcal{C}_2 = \mathcal{C}$. Since we want to apply balanced partitions where the number of robots in one component, $|\mathcal{C}_1|$, is close to the number of robots in the other component, $|\mathcal{C}_2|$, we define a *balance factor* of a partition $(\mathcal{C}_i, \mathcal{C}_j)$ using the function

$$f(\mathcal{C}_i, \mathcal{C}_j) = |\mathcal{C}_i|\,|\mathcal{C}_j|.$$

This function f is maximized when the number of elements in $\mathcal{C}_i$ is equal to the number of elements in $\mathcal{C}_j$, since all partitions satisfies the constraint $|\mathcal{C}_i|+|\mathcal{C}_j| = |\mathcal{C}|$.

We define the *assembly tree* as a directed binary tree, $\mathcal{T} = (\mathcal{V}^\mathcal{T}, \mathcal{E}^\mathcal{T})$, where the vertices $\mathcal{V}^\mathcal{T}$ are the components to be assembled, and the set of relationships between components and subcomponents is represented by the edges, $\mathcal{E}^\mathcal{T}$. In this hierarchy each component has two subcomponents, except for the root nodes which are isolated robots. The binary tree is initialized with the vertex of the complete structure $\mathcal{V}^\mathcal{T} = \{\mathcal{C}\}$, $\mathcal{C} = \{1, ..., n\}$ and no edges, $\mathcal{E}^\mathcal{T} = \emptyset$.

By definition of *feasible structure*, we know that the structure can be decomposed into a sequence of dockable parts. However, we do not know the valid sequence of partitions. In the worst case, where the n modules are in a line formation, there are $n - 1$ ways to decompose the set of robots into two subsets. Applying the same logic for the sub-partitions, a structure with n modules can be subdivided into $(n-1)!$ possible partitions. Therefore, an algorithm to check all these possibilities would have time complexity $\mathcal{O}(n!)$. Initially, we will present a centralized version which reduces the time complexity by using a dynamic programming approach.

Algorithm 1 describes a recursive method to compute an assembly tree giving priority to balanced partitions. In

Algorithm 1: C-ComputeAssemblyTree($\mathcal{C}, \mathcal{Z}, M$)

1 **if** $M[\mathcal{C}] \neq \emptyset$ **then**
2 **return** $M[\mathcal{C}]$ ▷ get stored tree for $\mathcal{C}$
3 **if** $|\mathcal{C}|= 1$ **then**
4 **return** $(\mathcal{C}, \emptyset)$ ▷ Minimum assembly tree
5 $\mathcal{P} =$ AllPossiblePartitions$(\mathcal{C}, \mathcal{Z})$
6 $\mathcal{P}' =$ SortByBalance$(\mathcal{P}, f)$
7 **foreach** $(\mathcal{C}_1, \mathcal{C}_2) \in \mathcal{P}'$ **do**
8 $\mathcal{T}_1 =$ C-ComputeAssemblyTree$(\mathcal{C}_1, \mathcal{Z}, M)$
9 $\mathcal{T}_2 =$ C-ComputeAssemblyTree$(\mathcal{C}_2, \mathcal{Z}, M)$
10 **if** $\mathcal{T}_1 \neq \emptyset$ **and** $\mathcal{T}_2 \neq \emptyset$ **then**
11 $\mathcal{V}^{\mathcal{T}} = \{\mathcal{C}\} \cup, \mathcal{V}_1^{\mathcal{T}} \cup \mathcal{V}_2^{\mathcal{T}}$
12 $\mathcal{E}^{\mathcal{T}} = \{(\mathcal{C}, \mathcal{C}_1), (\mathcal{C}, \mathcal{C}_2)\} \cup \mathcal{E}_1^{\mathcal{T}} \cup \mathcal{E}_2^{\mathcal{T}}$
13 $M[\mathcal{C}] = (\mathcal{V}^{\mathcal{T}}, \mathcal{E}^{\mathcal{T}})$ ▷ Store computed tree
14 **return** $(\mathcal{V}^{\mathcal{T}}, \mathcal{E}^{\mathcal{T}})$
15 **return** $\emptyset$

Algorithm 2: D-ComputeAssemblyTree($i, \mathcal{C}, \mathcal{Z}, M$)

1 **if** $M[\mathcal{C}] \neq \emptyset$ **then**
2 **return** $M[\mathcal{C}]$ ▷ get stored tree for $\mathcal{C}$
3 **if** $|\mathcal{C}|= 1$ **then**
4 **return** $(\mathcal{C}, \emptyset)$ ▷ Minimum assembly tree
5 $\mathcal{P} =$ AllPossiblePartitions$(\mathcal{C}, \mathcal{Z})$
6 $\mathcal{P}' =$ SortByBalance$(\mathcal{P}, f)$
7 **foreach** $(\mathcal{C}_1, \mathcal{C}_2) \in \mathcal{P}'$ **do**
8 **if** $i \notin \mathcal{C}_1$ **then**
9 $\mathcal{C}_1, \mathcal{C}_2 = \mathcal{C}_2, \mathcal{C}_1$ ▷ Robot i belongs to $\mathcal{C}_1$
10 $\mathcal{T}_1 =$ D-ComputeAssemblyTree$(i, \mathcal{C}_1, \mathcal{Z}, M)$
11 $b =$ isValidPartition$(\mathcal{C}_2)$ ▷ Ask to any robot in $\mathcal{C}_2$
12 **if** $\mathcal{T}_1 \neq \emptyset$ **and** b **then**
13 $\mathcal{V}^{\mathcal{T}} = \{\mathcal{C}, \mathcal{C}_2\} \cup, \mathcal{V}_1^{\mathcal{T}}$
14 $\mathcal{E}^{\mathcal{T}} = \{(\mathcal{C}, \mathcal{C}_1), (\mathcal{C}, \mathcal{C}_2)\} \cup \mathcal{E}_1^{\mathcal{T}}$
15 $M[\mathcal{C}] = (\mathcal{V}^{\mathcal{T}}, \mathcal{E}^{\mathcal{T}})$
16 **return** $(\mathcal{V}^{\mathcal{T}}, \mathcal{E}^{\mathcal{T}})$
17 **return** $\emptyset$

order to avoid recomputing the same substructures multiple times, we store each computed assembly tree in a variable M. Lines 1 and 2 check if the assembly tree of the input component $\mathcal{C}$ was already computed, if so, it returns the stored tree. The stop condition for the recursion is triggered when the input component is a single module (Lines 3 and 4). Line 5 computes all of the possible vertical and horizontal partitions for the points in component $\mathcal{C}$ (the maximum number of cuts is $|\mathcal{C}|-1$, which is the case when all of the robots are in a line formation). Since we want to give priority to the balanced partitions, Line 6 sorts the partitions using the balancing factor f. The for-each loop in Lines 7-14 checks if there exists a valid partition. Lines 8 and 9 recursively compute the assembly tree for each sub-component. We check if the partition is valid in Line 10, meaning that there are associated assembly trees for each partition. We store the computed assembly tree for the component $\mathcal{C}$ in Line 13 and return it in Line 14. The result of this centralized approach is a full assembly tree $\mathcal{T}$ with prioritized balanced partitions.

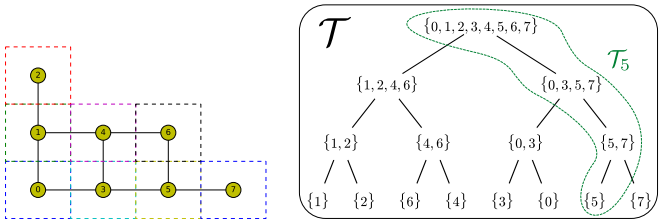
We highlight that this dynamic programming based approach reduces the time complexity from $\mathcal{O}(n!)$ to $\mathcal{O}(n^2)$ by saving the computed trees and avoiding recomputing. The maximum number of subcomponents with one module in a structure with n modules is n; the maximum number of subcomponents with two modules is $n-1$; and continuing the same logic until having a single subcomponent with n modules. Therefore, the total number of subcomponents is $\sum_i^n i = n(n+1)/2$, which demonstrates that the complexity is reduced to $\mathcal{O}(n^2)$. Taking into account the sorting method for balanced partitions costs $\mathcal{O}(n\log(n))$, and we are using it for each recursion, the complexity of Algorithm 1, is $\mathcal{O}(n^3\log(n))$.

In the decentralized version of the algorithm, each robot computes its own branch of the assembling tree $\mathcal{T}_i$ by only expanding the branch of the component that it belongs to. Instead of computing the other branches, the robot just ask if they are valid or not. After robot i executes Algorithm 2, it get its own assembly tree $\mathcal{T}_i$. Since all the robots follow the same priority based function, the union of all assembly trees will give the whole assembly tree $\mathcal{T} = \cup_i \mathcal{T}_i$. We can see that Algorithm 2 is similar to Algorithm 1, the main differences are the following. In Line 11, instead of computing the assembly tree for partition $\mathcal{C}_2$, robot i asks any robot in $\mathcal{C}_2$ if $\mathcal{C}_2$ is a valid partition or not. Line 13 and 14 only compute the branch of interest for robot i. We illustrate the resultant assembly tree for a desired structure in Figure 3.

Due to the nature of the binary tree, if all the partitions are balanced, the robots will be able to assemble the structure in time $\mathcal{O}(\log(n))$.

C. Stage 3: Performing Docking Actions

Once each robot i computes its own assembly tree $\mathcal{T}_i$ and is located in its assigned point in $\mathcal{Z}^E$, it proceeds to follow its sequence of docking actions, as described by Algorithm 3. In the beginning, robot i initializes a component with a



(a) Desired structure (b) Assembly tree $\mathcal{T}$

Fig. 3. A desired structure and its assembly tree. Panel (a) shows a desired planar structure from a top view, where the square modules are formed by the dashed lines and the allocated robot IDs are inside the disks. The continuous straight lines represent the connections between the robots. Panel (b) shows the total assembly tree $\mathcal{T} = \cup_i \mathcal{T}_i$, and the resultant assembly tree $\mathcal{T}_5$ for robot 5 by using Algorithm 2.

single element (Line 1). In Line 3, the component $\mathcal{C}_1$ uses the assembly tree to identify the next component that it is going to dock to. The *Sibling* function returns the node in the binary tree, $\mathcal{T}_i$, that has the same parent as the node that represents $\mathcal{C}_1$. Lines 4 and 5 compute the geodesic distance of each component to the centroid in $\mathcal{G}$. Since we want to attach all of the robots surrounding the centroid, z_c, we only move the component that is farthest from z_c. If the sibling component $\mathcal{C}_2$ is ready (it is in the right location and all its neighboring robots are properly attached) and is closer to the centroid, then component $\mathcal{C}_1$ moves towards $\mathcal{C}_2$ to make the attachment (Line 7). By the definition of a structured graph $\mathcal{G}$, we know that the graph $\mathcal{G}$ is connected. Therefore, there exists an edge $(p, q) \in \mathcal{E}$ such that $p \in \mathcal{C}_1$ and $q \in \mathcal{C}_2$. Then, the component $\mathcal{C}_1$ approaches $\mathcal{C}_2$ following the control law

$$u_i = K\,(\hat{x}_p - x_p),\ \forall i \in \mathcal{C}_1, \tag{2}$$

where $\hat{x}_p = x_q + z_p - z_q$ is the desired location for robot p with respect to robot q. The matrix of gains

$$K = \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix},$$

prioritizes the movement in a desired axis based on the parameters k_1 and k_2. We use these parameters to move robots toward one particular axis of attachment faster than the other axis. For example, if the attachment is vertical, we use $k_1 = 3k$ with $k_2 = k$, where $k > 0$ is a gain constant. Similarly, we would use $k_1 = k$ with $k_2 = 3k$ for the horizontal case.

In Line 9, $\mathcal{C}_1$ waits in its location while $\mathcal{C}_2$ gets ready, or meanwhile $\mathcal{C}_2$ performs its docking action. If a new attachment is created between two siblings, they consider themselves a new single component in Line 11 and remove the separated parts in the nodes of the assembly tree $\mathcal{T}_i$. The process from Lines 2-12 is repeated until a single component is successfully assembled to complete the desired structure.

We can deal with delays during the parallelized docking actions. These delays can be caused by errors in the motion actuators or simply because some robots may move slightly faster than others. In this case, some components will not be assembled on time, the robots that complete their docking actions on time will wait for the delayed components to be ready before executing the next motion (in Line 9). In this way, our algorithm supports delays during the docking actions.

When the robots are located in $\mathcal{Z}^E$, each robot is surrounded by a larger square that we call a *cell*. Each cell delimits independent area, and adjacent cells have a common boundary. We illustrate the cells in Figure 4(a). During a docking action, component $\mathcal{C}_1$ moves towards component $\mathcal{C}_2$. We call *safe region* to the convex hull the cells that belongs to the implicated components $\mathcal{C}_1 \cup \mathcal{C}_2$. Within this safe region, robots can freely move and generate alignment actions without risk of collision. We illustrate safe regions during docking actions in Figures 4(b) and 4(c). We can see that the colored background regions define a separation gap between any pair of robot paths.

We use the concepts of *cells* and *safe regions* to show the correctness of Algorithm 3 in the following proposition.

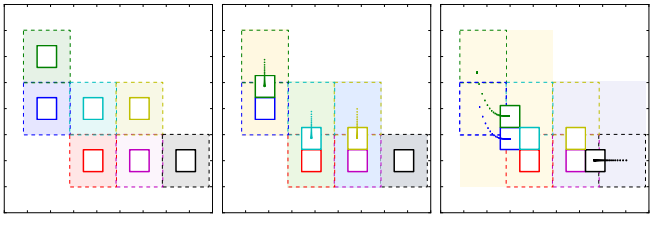
Proposition 1. *If all the robots in locations $\mathcal{Z}^E$ follow Algorithm 3, the complete desired structure is achieved without undesired collisions nor undesired attachments.*

Proof. As a premise, the robots are located in $\mathcal{Z}^E$ before starting the algorithm. Then, each of them has an associated independent cell. In the first iteration of the loop (Lines 2-16), each pair of siblings in the assembly tree performs a docking action at the same time. Lets say that a pair of robots is $\mathcal{C}_1 = \{i\}$ and $\mathcal{C}_2 = \{j\}$, then robot i moves towards the cell of robot j (as described by Line 7). Equation (2) drives robot i from its location, x_i, to its desired docking location $\hat{x}_i$. Then, robot i performs its docking action within its safe region, meanwhile robot j just keeps its location within its own cell. Since all robots perform docking actions within their own safe regions, and the safe regions are independent, all the docking actions performed in parallel are collision free. After this iteration, all of the robots remove the first level of the assembly tree (Line 12).

In the next iterations, we have a general component $\mathcal{C}_i$ that docks to its sibling $\mathcal{C}_j$ by following its assembly tree $\mathcal{T}_i$. A safe docking action is then performed by moving the robots

Algorithm 3: PerformDockingActions$(i, \mathcal{T}_i)$

```
1  C_1 = {i}
2  while |V^{T_i}| > 1 do
3      C_2 = Sibling(C_1, T_i)
4      d_1 = dist(C_1, z_c)          ▷ Distance to the centroid
5      d_2 = dist(C_2, z_c)
6      if ∃C_2 and d_1 > d_2 then
7          dockTo(C_1, C_2)          ▷ C_1 docks to C_2 with (2)
8      else
9          wait(C_1)                 ▷ Wait in place
10     if CheckAttachment(C_1, C_2) then
11         C_1 = C_1 ∪ C_2
12         V^{T_i} = V^{T_i} \ {C_1, C_2}
```



(a) (b) (c)

Fig. 4. Three steps of the assembly algorithm for a structure with seven robots. The robots are represented by the continuous-line-squares and their cells are represented by the dashed squares. The background colored areas represent the safe regions. Their trajectories at each time step are represented by the dots. Panel (a) depicts the robots in $\mathcal{Z}^E$ and their respective cells. Panel (b) and Panel (c) illustrate the first and the second docking actions, respectively.

within their own safe region (see Figure 4(c)). When all of the robots complete this iteration, they remove another level of the assembly tree.

Continuing the iterative loop, each robot will follow its assembly tree $\mathcal{C}_i$ and remove one level at a time. Since all the robots are following their own assembly tree, and we know that $\mathcal{T} = \cup\mathcal{T}_i$, they follow the whole assembly tree $\mathcal{T}$ and stop when they become a single component.

□

D. Summary

We summarize PAA in Algorithm 4. Given a set of n modular robots in the Euclidean space $\mathbb{R}^2$ and a feasible structure, specified by $\mathcal{Z}$, and assuming the distance between every pair of robots is greater than $2w$ and that Assumption 2 is satisfied, each robot i can follow Algorithm 4 to assemble the structure.

The following proposition states the completeness of PAA.

Proposition 2. *Given a feasible structure $\mathcal{Z}$, if a team of $|\mathcal{Z}|$ modular robots in arbitrary locations satisfy the minimum separation distance $2w$ and follow PAA, then they will assemble to the given structure while avoiding collisions and unexpected attachments.*

Proof. In Line 1, each robot can compute $\mathcal{Z}^E$ based on Equation 1, since they receive the structure $\mathcal{Z}$.

By design, the robots satisfy the requirements to apply D-CAPT; they are holonomic, the distances between robots and destination points are greater than $2w$. By Theorem 5.2 [17], we can guarantee that the robots move from their arbitrary locations to $\mathcal{Z}^E$ (Line 2).

Based on their locations in $\mathcal{Z}^E$, the robots know their location in the final structure, and thus each of them can compute their own branch of the assembly tree $\mathcal{T}_i$ (Line 3). Finally, in Line 4, by Proposition 1, the robots will follow the assembly tree in order to build the given structure. □

IV. Experiments

In our experiments, we show that our method is scalable and is able to assemble any connected structure. Initially we present multiple simulations and then we validate that our algorithm can be deployed on actual robots.

A. Simulations

We simulated different configurations in order to show the generality of the algorithm. We present four different configurations in Figures 5(a)-5(d). We present a single rectangular landing platform, a fence, a complex structure with a hole in the middle, and the bridge from Figure 1. They have 12, 20, 58, and 62 robots, respectively. All structures successfully assembled as expected. In Figures 5(e)-5(f) we represent the trajectories. It is possible to see how the robots speed up to move towards the connection, and then slow-down to form the attachment. It can be seen that the attachments tend to be aligned horizontally or vertically.

The plots in Figures 5(i) to 5(l) show the evolution of the completed attachments as a function of time. As a consequence of the use of binary trees in the assembly task, we can see that the structure is completed in time $\mathcal{O}(\text{Log}(n))$, since these structures can be decomposed into balanced partitions. We tested with different shapes and increased the number of robots from 12 to 60, as a result, the assembly time was increased by only two time units. The shape of the structure does not affect the upper bound of the construction time, but it changes the number of attachments in a given time step. The best case is when all the robots are in a line, where the number of attachments follows a logarithmic function of time, i.e. at the first time step, $n/2$ robots will be attached, then $n/2 + n/4$ and so on.

Algorithm 4: PAA$(i, \mathcal{Z})$

1 $\mathcal{Z}^E = $ ComputeExpandedGoals$(\mathcal{Z})$
2 D-CAPT$(x_i, \mathcal{Z}^E)$
3 $\mathcal{T}_i = $ D-ComputeAssemblyTree$(i, \mathcal{V}, \mathcal{Z}, \{\})$
4 PerformDockingActions$(i, \mathcal{Z}_i)$

B. Experiments

In our experiments with actual robots, we designed a holonomic square modular robot based on the Crazyflie aerial vehicle platform [18]. The robot has four propellers which are oriented to generate thrust in the plane. In this way, the robot can move on surfaces with low friction, or adapted to maneuver on water. In order to determine the robot pose in space as well as relative locations for docking, we are using a motion capture system (VICON) operating at 30 Hz. All commands are computed in ROS and sent to the robot via radio at 2.4 GHz. The docking mechanism is based on passive actuators which in this case is composed of sixteen magnets located in the corners of a cuboid cage. In Figure 6, we show the robot and the configuration of its four propellers and motors. Based on this configuration, the robot kinematics are described by

$$\dot{x} = Av,$$

where the vector $\dot{x} \in \mathbb{R}^3$ represents the velocities along the x and y axes, and the robot's angular velocity. The vector $v = [v_1, v_2, v_3, v_4]$, $v_i \in \mathbb{R}_{\geq 0}$ represents the linear velocities generated by each actuator. The transformation matrix for this given configuration is defined by

$$A = \begin{bmatrix} \cos(\pi/4) & \cos(3\pi/4) & \cos(-3\pi/4) & \cos(-\pi/4) \\ \sin(\pi/4) & \sin(3\pi/4) & \sin(-\pi/4) & \sin(-3\pi/4) \\ -1/r & 1/r & -1/r & 1/r \end{bmatrix},$$

where r is the distance from the propellers to the center of the robot.

Since the actuators can only generate positive thrust, we need a method to compute feasible solutions where $v_i \geq 0$. Solving the linear system by computing the pseudo-inverse of matrix A does not always satisfy the constraints. For this reason, we model the problem as an optimization problem where we need to minimize the function

$$\min(\|\dot{x} - Av\|^2),$$
$$\text{s.t } v_i \geq 0.$$

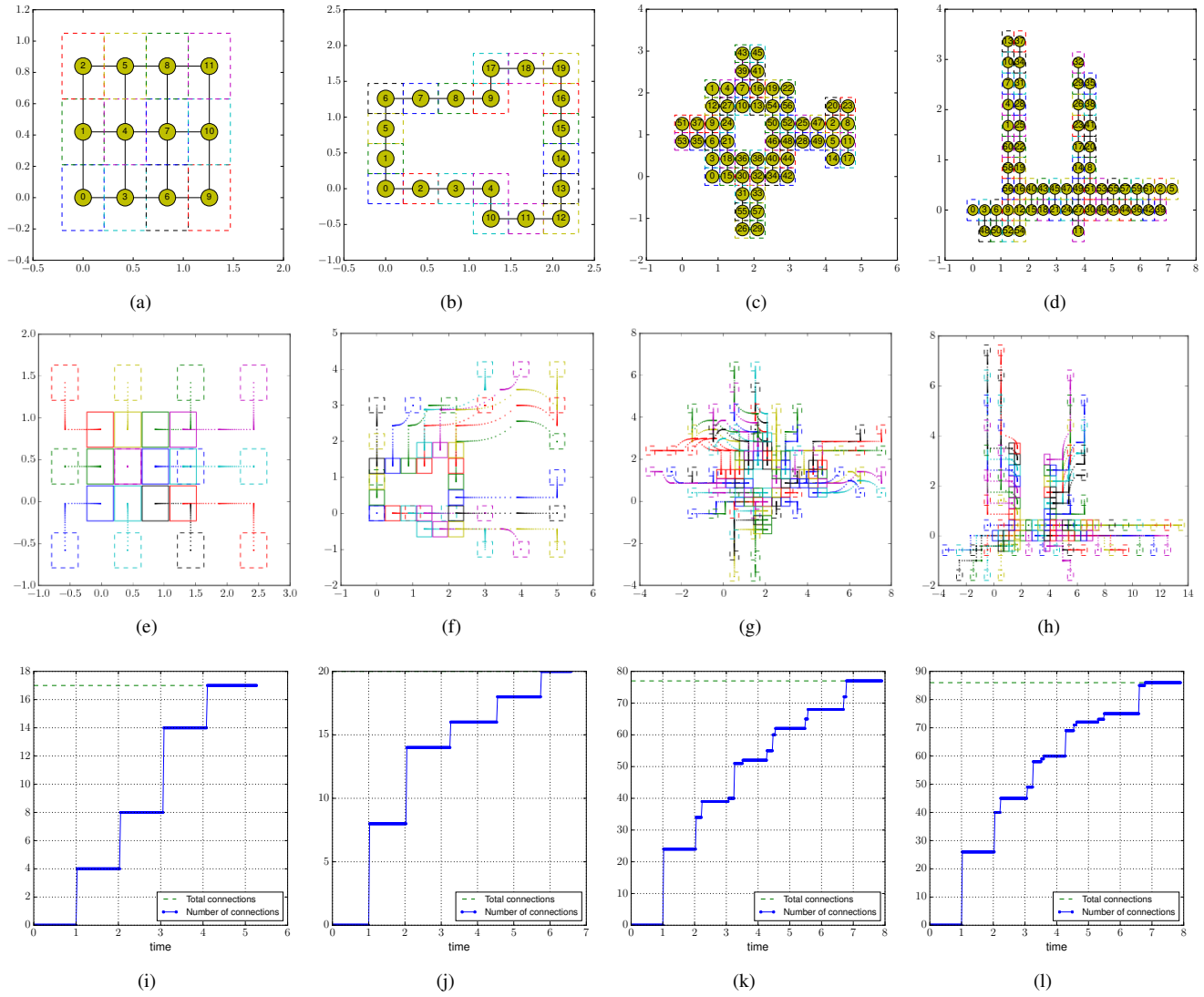


Fig. 5. Simulations for four different configurations. Panels (a)-(d) show the following structures: a landing platform, a fence, a complex shape with a hole in the center, and a bridge. The trajectories of the robots are presented in Panels (e)-(h), the dashed-line-boxes represent the robots in their expanded locations $\mathcal{Z}^E$, the continuous-line-boxes represent the their final destinations $\mathcal{Z}$, and the dots represent the robot locations on time. Panels (i)-(l) show the evolution of completed attachments on time.

We solve this optimization problem using the Sequential Least SQuares Programming optimization algorithm (SLSQP). In this way, we obtain feasible solutions for v given the desired velocity $\dot{x}$.

During the experiments, the robot performs well in both horizontal and vertical directions, which fits well with our method to align the robots during the docking action. A resultant configuration can be seen in Figure 7.

A factor that does not considerably affect the robot motion is the prop wash because the propellers that generate specific actions are sufficiently spaced apart. When a robot wants to move either forward or sideways, it only uses two motors for this action (e.g. it only uses v_1 and v_2 to move forward). For this reason, in the case when multiple robots are attached, the active propellers are separated by $\sqrt{2}w/2$.

In the attached video[1], we present simulated and actual robots to assembly multiple structures. We can see how the docking actions require time to align and move accurately.

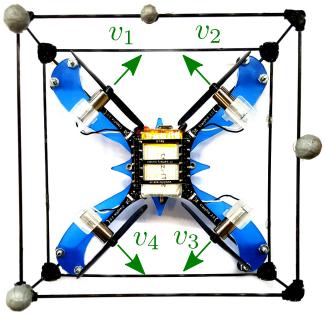


Fig. 6. Top view of the single holonomic modular robot that was used in the experiments. The green arrows represent the velocities that each motor is able to generate.

[1]Link: https://www.youtube.com/watch?v=r9UDHhCWRQw

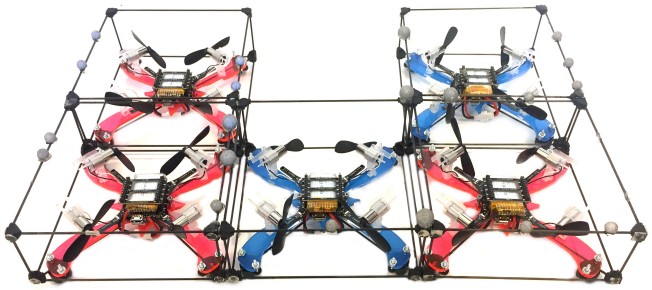


Fig. 7. Five holonomic modular robots in a U-configuration. The robots are attached by magnets on the corners of the carbon fiber cages.

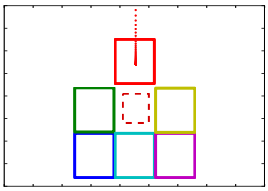


Fig. 8. Undesirable configuration to build a structure. The robots are represented by the continuous-line-squares and the missing spot is represented by the dashed line. In this U-shape, there is a spot that cannot be filled by the red robot.

Due to friction and errors in the actuators, the robot motion presents inaccurate translations and undesired rotations, but the magnetic force in the cages fix up to 1cm of motion error. Using our algorithm, each assembly process is completed, even when some robots move slower than others.

C. Discussion

An alternative assembly method would be to "shrink", based on the expanded formation generated in Stage 1. The disadvantage of this method is that robots would have to generate precise docking actions at exactly the right time or the structure would not be able to be completed. An example of this would be the case when robots attaching to form one side of a square platform do not simultaneously attach, and therefore a "U" shaped spot is created in the side of the structure. The robot assigned to this spot will no longer be able to dock to the structure, as noted in [1], we illustrate that case in Figure 8. The main difficulty is mainly due to multiple moving robots trying to dock at the same time. PAA eliminates the potential of this case occurring, since it is based on docking pairs of independently joined components.

V. Conclusions and Future Work

In this paper, we presented a decentralized algorithm to assemble structures with modular robots. Our approach drives the robots to autonomously assemble a structure in the planar space. Since docking actions are a bottleneck in the assembly process, our algorithm parallelizes these actions while avoiding collisions and undesired attachments. We show the generality of our algorithm using simulations. Through experiments with actual robots, we show that our algorithm can be deployed in real-settings.

The future directions of this work include the extension of our current approach to different robot shapes, such as hexagons, as well as different ways of splitting the components. In our algorithm, we have to expand the graph in order to create enough space, and to avoid undesired collisions. We plan to study how to attach the robots in environments with obstacles, where the robots cannot expand in specific directions.

References

[1] 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," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1234–1240, 2014.

[2] J. Seo, M. Yim, and V. Kumar, "Assembly sequence planning for constructing planar structures with rectangular modules," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 5477–5482.

[3] J. Werfel and R. Nagpal, "Three-dimensional construction with mobile robots and modular blocks," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 463–479, 2008. [Online]. Available: http://dx.doi.org/10.1177/0278364907084984

[4] L. Murray, J. Timmis, and A. Tyrrell, "Modular self-assembling and self-reconfiguring e-pucks," *Swarm Intelligence*, vol. 7, no. 2, pp. 83–113, 2013.

[5] E. Klavins, S. Burden, and N. Napp, "Optimal rules for programmed stochastic self-assembly," *Self*, vol. 6, p. 7, 2006.

[6] J. Werfel, K. Petersen, and R. Nagpal, "Designing collective behavior in a termite-inspired robot construction team," *Science*, vol. 343, no. 6172, pp. 754–758, 2014.

[7] L. Chaimowicz, N. Michael, and V. Kumar, "Controlling Swarms of Robots Using Interpolated Implicit Functions," *2005*, no. April, pp. 2487–2492, 2005.

[8] M. A. Hsieh, A. Cowley, J. F. Keller, L. Chaimowicz, B. Grocholsky, V. Kumar, C. J. Taylor, B. Jung, and D. C. Mackenzie, "Adaptive Teams of Autonomous Aerial and Ground Robots for Situational Awareness," vol. 24, no. August, pp. 991–1014, 2007.

[9] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013.

[10] E. Tuci, R. Groß, V. Trianni, F. Mondada, M. Bonani, and M. Dorigo, "Cooperation through self-assembly in multi-robot systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 115–150, 2006.

[11] L. Cucu, M. Rubenstein, and R. Nagpal, "Towards self-assembled structures with mobile climbing robots," *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 1955–1961, 2015.

[12] J. W. Romanishin, K. Gilpin, and D. Rus, "M-Blocks : Momentum-driven , Magnetic Modular Robots," pp. 4288–4295, 2013.

[13] R. Oung and R. D'Andrea, "The distributed flight array," *Mechatronics*, vol. 21, no. 6, pp. 908–917, 2011.

[14] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of Cubic Structures with Quadrotor Teams," *Mechanical Engineering*, 2011.

[15] F. Augugliaro, S. Lupashin, M. Hamer, C. Male, M. Hehn, M. W. Mueller, J. S. Willmann, F. Gramazio, M. Kohler, and R. D'Andrea, "The flight assembled architecture installation: Cooperative contruction with flying machines," *IEEE Control Systems*, vol. 34, no. 4, pp. 46–64, 2014.

[16] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001.

[17] M. Turpin, N. Michael, and V. Kumar, "Concurrent assignment and planning of trajectories for large teams of interchangeable robots," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 842–848, 2013.

[18] W. Hoenig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian, "Mixed reality for robotics," in *IEEE/RSJ Intl Conf. Intelligent Robots and Systems*, Hamburg, Germany, Sept 2015, pp. 5382 – 5387.