Distributed Assembly with Online Workload Balancing and Visual Error Detection and Correction

James Worcester, M. Ani Hsieh, and Rolf Lakaemper

Abstract We consider the assembly of a three dimensional (3D) structure by a team of heterogeneous robots capable of online sensing and error correction during the assembly process. We build on our previous work and address the partitioning of the assembly task to maximize parallelization of the assembly process. Specifically, we consider 3D structures that can be assembled from a fixed collection of heterogeneous tiles that vary in shapes and sizes. Given a desired 3D structure, we first compute the partition of the assembly strategy into N_a sub-components that can be executed in parallel by a team of N_a assembly robots. The assembly robots then perform online workload balancing during construction to minimize assembly time. To enable online error detection and correction during the assembly process, mobile robots equipped with visual depth sensors are tasked to scan, identify, and track the state of the structure. The result is a cooperative assembly framework where assembly robots can balance their individual workloads online by trading assembly components while scanning robots detect and reassign missing assembly components online. We present the integration of the planning, sensing, and control strategies employed in our framework and report on the experimental validation of the strategy using our multi-robot testbed.

1 Introduction

Distributed autonomous assembly of general two (2D) and three dimensional (3D) structures is a complex task requiring robots to have the ability to: 1) sense and

James Worcester and M. Ani Hsieh
Drexel University, Mechanical Engineering & Mechanics, Philadelphia, PA, 19104, e-mail: `\{jbw68,mhsieh1\}@drexel.edu`

Rolf Lakaemper
Temple University, Computer & Information Sciences, Philadelphia, PA 19122 e-mail: `lakamper@temple.edu`

manipulate assembly components; 2) interact with the desired structure at all stages of the assembly process; 3) satisfy a variety of precedence constraints to ensure assembly correctness; and 4) ensure the stability and structural integrity of the desired structure throughout the assembly process. While the distributed assembly problem represents a class of tightly-coupled tasks that is of much interest in multi-robot systems (Chaimowicz et al, 2001), it is also highly relevant to the development of next generation intelligent, flexible, and adaptive manufacturing and automation.

The execution of tightly-coupled tasks by multi-robot teams has mostly focused on cooperative grasping and manipulation (Mataric et al, 1995; Fink et al, 2008). These works, however, do not address the challenges imposed by the need to satisfy specific precedence constraints during assembly process. These constraints are especially important in applications like automated palletizing, construction, manufacturing, infrastructure repair and maintenance since automated strategies must ensure correctness as well as stability of the resulting structures. While there has been significant focus in micro/nano-scale assembly (Klavins, 2007; Evans et al, 2010; Matthey et al, 2009; Rai et al, 2011), automated macro-scale assembly is gaining increased attention. Recent work in this area includes (Petersen et al, 2011; Yun et al, 2009; Yun and Rus, 2010; Stein et al, 2011; Heger and Singh, 2010; Lindsey and Kumar, 2012).

In Werfel and Nagpal (2008) and Petersen et al (2011), assembly is achieved through a combination of robots with limited sensing and actuation capabilities and assembly components capable of storing and communicating location information with the robots. The focus of these works is on designing a set of consistent local attachment rules that ensure completeness and correctness of the assembly, while obeying local constraints between pieces and avoiding unrecoverable situations. In Yun et al (2009); Yun and Rus (2010); Stein et al (2011), a workload partitioning strategy is presented to enable a team of robots to achieve parallel construction at the macro scale. The approach maintains a Voronoi decomposition of the structure based on the assembly robots' locations by minimizing the total difference in the masses of the assembly components in each cell. Failures of robots, ordering constraints, and changes to an existing structure are also addressed.

Despite these successes, significant challenges still remain. First, existing macro-scale assembly strategies often reduces to a serialization of the assembly procedure despite employing multiple robots (Petersen et al, 2011; Lindsey and Kumar, 2012). While this ensures correctness of the resulting structure and safe execution, it can significantly hamper productivity by not exploiting parallelization in the assembly process. Second, existing strategies often rely on external sensors for localizing the assembly components (Lindsey and Kumar, 2012; Yun and Rus, 2010), *i.e.*, stationary sensors mounted in the workspace. While such a strategy may be feasible for small work cell volumes, such an approach may be challenging for large workspaces since it would be difficult to provide enough coverage and accurate localization.

In this work, we present a cooperative assembly strategy where the objective is to coordinate a heterogeneous team of robots to collaboratively assembly a specific class of 3D structures. Specifically, we consider the partitioning of the heterogeneous robot team into assembly and scanning robots. Assembly robots will be tasked to assembly the desired 3D structure using a collection of assembly components of varying shapes and sizes. Building on our previous work Worcester et al (2011), we determine an allocation of the assembly task into subcomponents to enable parallel assembly by the assembly robots. This planning phase minimizes the workload imbalance between the assembly robots without violating local attachment constraints, given by the geometry of assembly tiles/components, and global precedence constraints, required for structural stability. While this partitioning strategy ensures the correctness of the distributed assembly strategy, the allocation is performed *a priori* and cannot cope with execution time assembly errors, *e.g.*, incorrect and/or missed placements. In this paper, we extend our existing work Worcester et al (2011) to enable online error detection and correction during the assembly process. This is aided by a small number of scanning robots capable of providing real-time visual feedback of the state of the structure during assembly. Additionally, we also include online workload balancing that do not violate local precedence constraints between assembly components. An advantage of the proposed workload balancing strategy is that it can be used with any pre-existing assembly plan modeled as a tree where the root node represents an assembly component located on the exterior of the desired structure. Our main contribution is cooperative assembly framework that integrates the planning, sensing, and workload balancing into a single coordination architecture for teams of heterogeneous robots.

The paper is organized as follows: We describe our methodology in Section 3. The experimental setup and results are presented in Section 6. The experimental insights and lessons learned are reported in Section 7. We conclude with directions for future work in Section 8.

2 PROBLEM STATEMENT

In this work, we build on the results in Worcester et al (2011) to devise an online workload balancing strategy for the distributed assembly problem. The objective will be to parition the task into subtasks as evenly divided as possible, so that each robot has close to the same amount of work. In addition to the preplanning algorithm from Worcester et al (2011), we add an online algorithm to enable robots to trade tasks for faster completion. A scanning robot equipped with a Microsoft Xbox Kinect is added to provide online error detection, so any missing parts can be detected and replaced. ~~which causes the assembly robots to replace the missing part.~~ We assume a team of N_a assembly robots, each capable of transporting a single assembly component from a cache location to the assembly site, and N_s scanning robots, equipped with visual depth sensors. Let M denote the number of distinct assembly components/tiles/nodes where t_i denotes a component/tile/node of type i. We will assume that each tile of type i can be described as a general polytope and that the robots know the geometries of the different tile types a priori. Furthermore, every tile of type i will have a fixed number of attachment sites. These attachment sites are locations where tiles can mate and lock onto other tiles. Let $\mathscr{W}$ denote the workspace and S_d denote a desired target structure. The structure-free portion of $\mathscr{W}$ is given by $\mathscr{W}_f = \mathscr{W} \setminus S_d$. Let $G_{S_d} = \{V_{S_d}, E_{S_d}\}$ denote the *structure graph* where each node in V_{S_d} represents each structural component that can be placed next to or on top of other component(s) by a single robot to form larger structures. An edge (u,v) exists in E_{S_d} if v can be reached from u through a path in $\mathscr{W}_f$ and vice versa. For every $(u,v) \in E_{S_d}$, we assign a weight equal to the planar Euclidean distance between u and v. The *mass* of a node is the amount of time required to build the node. We also assume that S_d is finite in size or V_{S_d} is a finite set.

We assume assembly constraints of the form $u \prec v$, or u must be built before v. This can represent a variety of assembly constraints, *e.g.*, different materials that must be combined in a specific sequence or the placement order of supporting components for structural stability during assembly. For a desired S_d, we define a *constraint graph* as a directed graph $G_C = \{V_C, E_C\}$ such that $V_C = V_{S_d}$ and $E_C = \{(u,v) | u \prec v\}$ and for every $(u,v) \in E_C$, we refer to u as a *support*, and to v as a *supported node*. In general, given S_d, these constraints can be obtained by adding an edge for each node that is directly supported by another node (*i.e.,* be placed after that node), derived from an AND/OR graph representation (Sanderson et al, 1990), or derived from a sequential assembly plan similar to Grushin and Reggia (2008). Finally, we define a directed graph $G_R = \{V_R, E_R\}$, *i.e.*, the *route graph*, where $V_R = V_{S_d}$ and E_R is given by $E_R = \{E_{S_d} \setminus E_D\}$ with $E_D = \{(u,v) | \, (v,u) \in E_C\}$. The route graph represents all the viable paths within the structure that do not travel from a supported node to its support. We assume an obstacle-free and fully connected $\mathscr{W}$ prior to the assembly of S_d to simplify the motion-plans used to estimate the time to assembly for individual components and to ensure all components in S_d are reachable at some point in the assembly process.

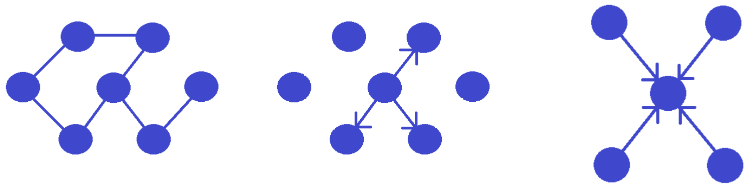

(a) (b) (c)

Fig. 1 (a) A structure graph, navigation between nodes is only possible if an edge is present between those nodes. (b) A constraint graph requiring that the central node be built before the adjacent nodes. (c) An example of a structure that would not be in the set of admissible structures S_A. The displayed constraints state that each of the outer nodes must be placed before the central node. Assuming the robot cannot navigate between two of the corner pieces, a robot cannot reach the central node once all of its supports have been placed.

We define the set of admissible structures S_A as structures for which G_R is strongly connected and G_C has no cycles. For 3-D structures, the limitations of the mobile manipulator adds the restriction that to be a member of S_A, each node must be reachable from $\mathscr{W}_f$ according to the geometry of the assembly robot. For each robot we define two neighbor sets. The first, N_1, is a static set of robots that the robot is allowed to make trades with. This is chosen to be the two robots with neighboring entrance nodes as defined by mapping the direction from the center to the entrance nodes onto a unit circle. The second, N_2, is a variable set encompassing all robots responsible for components adjacent to that robot's own components. This set N_2 is the set of robots that may place a support node that affects the current robot. We will often use the term *neighbors* to describe the set N_1.

Finally, we assume robots are able to localize within the workspace, identify and manipulate the components located at the parts/components cache, and identify the structure throughout assembly. While our work is focused on ground mobile manipulators, the proposed partitioning strategies can be extended to other autonomous robots.

The objective can then be stated as requiring the ~~is for the~~ N_a assembly robots to build the desired structure S_d, which must be a member of S_A, without violating the constraint graph G_C. During this process, a balanced workload will be maintained via the online trading algorithm and any errors will be reported by the N_s scanning robots to be corrected by the assembly robots. All communications for a robot must be limited to the sets N_1 and N_2 defined for that robot.

3 Methodology

3.1 Task Partitioning

Given S_d and N_a assembly robots, we employ the approach described in Worcester et al (2011) to determine an appropriate partitioning of the assembly of S_d into N_a tasks that can be executed in parallel. The objective is to arrive at a partition that maximizes parallel execution of the assembly while minimizing workload imbalance between the robots without violating any of the placement precedence constraints between the assembly components. The approach uses Dijkstra's algorithm with multiple starting nodes to generate a set of assembly tasks for each robot. Rather than each node finding the shortest path to a single start node, this results in finding the shortest path to any start node. This results in a partitioning of components of S_d such that each robot's task is composed of tiles that are closest to its starting node, and each task is represented by a tree. By building from the leaves back to the root it is possible to avoid blocking access to unbuilt nodes within the task. The only limitation on this is that we do not allow robots to claim a supporting node as a child of the node it supports. This allows the robot to build from leaves to root without violating constraints on order. The starting nodes are chosen to be equally spaced along the exterior. The output of this part will look something like Fig. 2. This initial allocation strategy is then improved with a second phase of node trading to yield a more balanced workload among the robots. The last step of this approach is the generation of an assembly sequence for each robot that minimizes the time a robot must wait for the placement of supporting tile by another robot. This is achieved by maximizing the time between a placement and the placements of any supporting tiles.

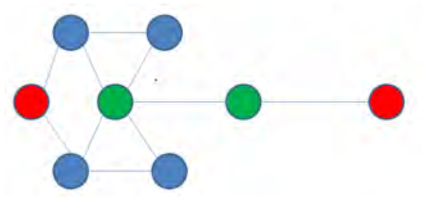


Fig. 2 An example of Dijkstra's algorithm with multiple roots applied to a graph. The red nodes represent the roots of the two trees. One robot claims the green nodes, the other claims the blue. The green robot cannot claim any more nodes because it has distance 3 to all the blue nodes, while the blue robot has distance at most 2 to any of its nodes.

It is important to note that the approach described in Worcester et al (2011) is a partitioning strategy that is executed *a priori* and generates a distributed assembly strategy for a team of N_a robots given S_d, and $\{t_1, \ldots, t_M\}$.

3.2 Online Workload Balancing

In this work we extend the preplanning approach described above to include online workload balancing, accomplished by having each robot independently propose appropriate trades of tasks. The preplanning approach generates a starting plan represented by a set of tree structures, one for each robot's subtask, with the restriction that the root node provides an exit from the structure. Each robot begins with full knowledge of the preplan, but may be unaware of changes made to that plan by other robots. The overall strategy used is to maintain accurate knowledge of N_1, which is the set of robots we are able to trade with. Outside this set, a robot has no knowledge of changes made to the plan unless they affect a robot in its N_1 set. These changes will be relayed by that robot, so no communication with robots outside N_2 is required. The only communications coming from outside N_1 are messages indicating that a support has been placed for an unbuilt node held by the robot, which necessarily comes from a robot in N_2. This class of messages could be replaced with a sensor capable of determining whether a specific support is present for a node that is otherwise ready to be built. By maintaining accurate knowledge of robots that can be trading partners, each robot is able to independently plan and propose beneficial trades. A robot will propose a trade whenever it is idle due to none of its task being currently buildable, or when its workload is below the average of its neighbors by at least twice the average build time for a node.

3.2.1 Node Trading Algorithm

The algorithm is based on Phase II of the preplanning algorithm, which is described in Worcester et al (2011). This approach considers three criteria for evaluating trades. These are the distance from the root to the node being considered, the difference in task size between the giving and receiving robot, and whether taking the node being considered would put both parts of a constraint in the same task (which reduces the dependency between robots). By applying these criteria to all neighbors of the current task, a robot can find the highest value trade it could currently make. For this application the algorithm has been revised to work in a distributed fashion, with each robot managing its own subtask and holding as accurate a representation as possible of neighbors' subtasks. The method a robot uses to manage its own task is to maintain a min-heap structure containing all nodes that are ready to be built. For each node that has not yet been added to the heap, it maintains two variables describing the number of unbuilt children in the tree structure representation of the task, and the number of unbuilt supports (nodes that must be built before that node, generally because they provide an immediate physical support). After building a node itself or receiving a message from another robot building a node, the algorithm updates both of these variables accordingly. When both have a value of zero, indicating that the robot can build the node without blocking its access to other parts of the structure and all supports are in place, it adds that node to the heap. The heap priority is described by:

$$P = d_{ci} - c_{ki} + (w * t_{si}) \tag{1}$$

where P is the heap priority, d_{ci} is the distance from the center of the structure to node i, t_{si} is the timestamp the last support was built for node i by another robot, and w is a weighting factor. The variable c_{ki} represents the benefit from building this node in terms of how many nodes it supports. The t_{si} factor gives a higher score, which corresponds to a lower priority, to nodes with supports that were placed more recently. During this process, the robot frequently checks for incoming messages, and sends a message to all robots in set N_1 whenever it builds a node.

3.2.2 Communications Protocol

This section details the types of messages, when they are sent and how they are dealt with. Messages are passed with the following information: *destination robot, message type, source robot,* and *data*. The *destination robot* field is used as a tag for what robot should pick up this message, with two exceptions that will be discussed later. The *source robot* is the robot sending the message (necessary in order to acknowledge receipt of messages), and the *data* field is of variable length depending on the type of message. The *message type* is interpreted as follows:

1. Sending robot built a node (data length 1)
2. Request to take a list of nodes (data length variable, first element gives length)
3. Answer to trade request (data length 1, yes or no)

4. Receipt of a new node along with the parent node it was attached to (data length 2, node and parent)
5. Sent a list of nodes to another robot (data length variable, specified by first element)
6. Built a node which acts as a support for a node held by receiving robot (data length 2, built node and supported node)
7. Acknowledgement of received message
8. Request for a resend of message type 1 if a certain node is built (data length 1)

Message types 2 through 5 are only sent to robots in the set N_1, while message types 1, 6, and 8 can also be sent to robots in the broader set N_2.

Throughout most of the task, these messages are aimed at a single robot. There are two exceptions which use a broadcast architecture, these are the coordination of the start and end of the experiment. In these cases we use special flags in place of the *destination robot* field, which indicate a different message structure then the norm. For the start case, there is no more information in the message, all robots start building as soon as they receive the message. For the end case, the rest of the message consists of a list of robots we know have finished. A robot done with its own task will only replace this message if they can add to its length (generally by adding their own id). When the list contains all IDs the robots stop. Coordination of finish times is necessary in order to be able to continue responding to requests for build confirmations until all tasks have been completed. Both of these message types are relayed by each robot. This means that communication only needs to be possible with robots in N_2 in order for successful completion of an assembly task.

When a robot receives a type 1 (node built) message, it updates its representation of the sending robot's task, and checks whether the built node acts as a support for any of its own nodes. For type 2 (trade request), the robot checks that 1) it is the current owner of the requested nodes, 2) the nodes are not built, and 3) it is currently not building any of the requested nodes. If all of these conditions are satisfied, the robot responds with a message of type 3 saying 'yes', sends a message of type 5 to N_1 listing the nodes lost, and updates its representation of its own task. The robot, however, does not yet update its neighbor's task because it does not know where the new nodes will be attached. If the conditions are violated, it instead replies with a message of type 3 saying 'no'. For type 3 (answer to trade), if the answer is no, the robot cancels the proposed trade and returns to the main routine. If the answer is yes, the robot modifies its representations of both its own task and its neighbor's task, and then sends a message of type 4 to N_1, detailing where on the tree each of the new nodes is added. Upon receipt of a type 4 or type 5 message, robots modify their representation of the sending robot's task.

The last two message types usually come from robots in the set $N_2 \setminus (N_1 \cap N_2)$, and deal with supporting nodes. Message type 6 informs a robot that a support has been built for one of its nodes. When receiving this message, the robot first checks whether it still holds that node. If it has traded that node, the sending robot may not be aware of that, being outside N_1. If it has that node, it modifies its own representation and decrements the variable representing the number of unbuilt supports. If it does not, it forwards the message to whichever robot it traded the node to, which it knows because it is maintaining a full representation of robots in N_1. The other message is type 8, which is a request for confirmation that a certain node has been built. This message is sent when a robot has an empty heap and can find nothing to take from its neighbors, but still has nodes with unbuilt supports. In this situation, a robot may have missed the message informing it that the support had been built and will request that message to be resent. When receiving a message of type 8, a robot checks whether the node being asked about was actually built. If so, the robot sends a message of type 1 to the requesting robot. If not, it does nothing.

Before building each node, a robot compares its own remaining task size with the average task size in its local neighborhood, which we define as the set N_1 plus itself. If the robot's own task size is below the average by at least twice the average node build time, it searches among its representation of its neighbors for a valid trade, which it then requests. The other time it looks for a trade is when its own build heap is empty, indicating that it will be idle if it cannot find work to take from another robot. The benefit of looking for trades even when it still has its own work to do is that earlier trading allows more flexibility in what nodes are exchanged, because less of the structure has been built. Once the robot decides to look for a trade, it scores the possible trades according to the criteria laid out in Worcester et al (2011), considering relative task sizes, relative distances to the node in question, and the benefit a robot gets from holding both parts of a constraint itself. We reward trades that bring both parts of a constraint to a single robot to minimize the amount of cross-robot interaction. While the assembly robots are doing this, the scanning robots inspect the structure to discover missed placements, as described below.

3.3 Complexity

It is possible to determine bounds on the length of the experiment as well as the number of messages sent, which are provided in the following two theorems.

Theorem 1. *The length of the experiment will be $O(M * D)$, where M is the number of tiles and D is the maximum distance to the cache.*

Proof. The only situation in which a robot will be idle while it has a non-empty task is if its remaining tasks have unsatisfied constraints. Because we assumed that the constraint graph may not have cycles, there must be at least one unbuilt tile which does not have an unsatisfied constraint. Therefore, at least one robot, the robot possessing that tile, will still be working. The time to deliver a single tile will be twice the time taken to drive to the cache combined with a constant amount of time for pickup and placement. This time is $O(D)$. The worst case scenario would be a chain of constraints such that only one tile can be built at a time. In this case the time taken will be $O(D)$ times the number of tiles M, giving a total time of $O(M * D)$.

Theorem 2. *With the exception of message type 8 and responses to it, the total number of messages sent will be $O(M * N_a)$.*

Proof. Message types 1 and 6 will only be sent once per tile, except as a response to message type 8, and are thus directly bound by $O(M)$. Messages 2 through 5 are all bound by the number of trades. There are two cases in which tiles are traded, either when a robot is idle or when a robot has less work than its neighbors. If a robot is idle, it will immediately build any tile it receives in a trade, which prevents that tile from being retraded, since a robot cannot give away a tile being built. This limits the number of trades due to idle robots to $O(M)$, as each such trade results in a tile being built. A robot taking work from its neighbors cannot take more than $M/2$ tiles unless it is then giving those tiles away, since this would necessarily involve taking from a neighbor with fewer tiles. If it is giving those tiles away, this pattern can only be repeated N_a times before arriving back at the original robot. Since each successive robot has to have fewer tiles, it is not possible for this chain to repeat, as the first robot cannot have fewer tiles than itself. Therefore the total number of trades is $O(M * N_a)$, which limits message types 2 through 5. Message type 7 is sent once for each other message, and therefore increases the number of messages by a constant factor of two, which does not change the complexity. It is not possible to limit the number of messages of type 8, as this will request message type 1 to be resent until it is successfully received.

3.4 Visual Feedback

To provide information to the robots about the current state of the physical structure S_p as it is being assembled, we implement a feedback system using visual depth sensors. The objective is to use online sensing to compare S_p with the robot's internal model of the currently assembled structure S_a, and to provide control data to the building process, based on differences between S_p and S_a. To keep an updated representation of the state of S_p, we add a sensing robot to the system, which is equipped with a depth sensor K_p, in our case the Microsoft Xbox Kinect sensor. The robot constitutes the system for visual inspection (VI). The VI-robot is independent of construction robots. It runs a prioritized exploration algorithm, which aims to map and update the dynamically changing physical structure with priority on currently targeted building regions. The input to this system is the internal structure S_a (Figure 3(b)), which models what the system is expected to see from the physical structure S_p (Figure 3(a)), and the raw visual sensor data (3D point cloud, Figure 3(c)), the output is a state for every block t_i of the internal structure S_a, denoting if the tile is present, missing, or occluded (currently no visual information about the tile is available).

Before an assembly robot adds a part t_i to the physical structure S_p, it queries the VI-system, if the targeted region data is updated and S_p matches the expected state of S_a. For this comparison, we simulate a robot internal system containing S_a and a virtual Kinect sensor K_v. Using ray-tracing, we simulate a Kinect scan of S_a. The outcome of the simulated ray tracing is compared with the real scan of the physical

Kinect K_p to compute the state for each tile $t_i \in S_a$. The following sections will explain the VI system in more detail.

3.4.1 Coordinate System Matching

To compare the outcome of the physical scan and the virtual scan, we must find P_v, the pose of K_v in the virtual system. If we let P_p denote the pose of K_p in the physical system, then P_v has to equal P_p. The positioning is performed in multiple steps consisting of an overhead localization system, a floor based correction, and an Iterative Closest Point (ICP) alignment. In the following we use a right handed coordinate system. The horizontal plane is described by (x,z), height is described by the y-axis.

First, an overhead localization system gives an estimate of the horizontal (x,z) position of K_p. This includes the (x,z) coordinates as well as the yaw α, i.e. the rotation angle around the y-axis. The overhead localization system is provided by a network of cameras with errors in (x,y) below 5 cm and angular errors in α of ¡10 degrees.

To complete P_p, the missing pose-parameters y (the Kinect's height) and β,γ (pitch and roll, i.e. rotation around x and z axis respectively) are determined by a floor-based correction. We perform floor detection in the point cloud C_p resulting from the physical scan. Since the floor in the physical system defines the $x-z$ plane, a transformation T_f, which aligns the floor's normal with the y axis of the virtual system completes the estimate of P_p. We compute T_f by regression of the floor points to their projections in the $x-z$ plane (point to plane correspondence). As such, we note that T_f has no y-rotation component, i.e. the Kinect's yaw, as previously determined by the overhead system, is not altered by an otherwise ambiguous rotation. In addition, we re-compute the translational part of T_f in the $x-z$ plane, such that only the vertical position of K_v is affected.

While there are errors in the localization and ground plan position estimates and noise in C_p, they provide a sufficiently good starting point for an Iterative Closest Point (ICP) alignment (Besl and McKay, 1992). We use P_p as a starting estimate for P_v, therewith we also transform C_p: we set $C_p \leftarrow T_f C_p$. We perform a 6D (3 location parameters, 3 directional parameters) point to plane ICP, with the goal to align C_p to S_a. ICP is a well known technique in robotics and computer vision, successfully applied to align (3D) point clouds, especially for robot mapping (Nüchter et al, 2005). Given two point clouds C_1 and C_2, it finds, in an iterative way, a (locally) optimal transformation T_c that minimizes the squared sum of distances between points in C_1 and their iteratively re-determined closest neighbors in $T_c C_2$. ICP is known robust and fast as long as a good starting estimate of the point-poses is provided. In practice, the previously described steps to compute P_p proved to be sufficient as a starting point.

We perform a fast point-to-plane ICP version: $C_1 \subset C_p$ originates from the physical scan C_p, and consists of a subset of points, being candidates for points belonging to S_p. C_2 is iteratively generated as the projection points of C_1 onto the virtual structure S_a. Internally, S_a is represented as a set of planar patches, describing the geometry of the tiles t_i. Storing the tiles of S_a together with a hierarchy of axis aligned bounding boxes (AABB) allows for fast computation of the projections of C_1 onto S_a. The hierarchy is given naturally: we store an AABB for the structure S_a, for each tile $t_i \in S_a$ and each planar patch $p \in t_i$. Using this hierarchy of bounding boxes, C_1 results in a relatively small subset of C_p. In addition, we omit points that belong to the floor, as determined by the floor detection step. A single Kinect scan in hi-res (640 x 480) contains about 300000 points, the typical point cloud of candidates describing reflections from the structure S_p, after filtering, typically reduces the number of points to less than 10000. We limit our ICP to a maximum of 10 iterations. ICP results in T_{ICP}, an accumulated rotation and translation to align C_1 to S_a. When we apply T_{ICP} to P_p, this reduces pose errors from the former computation. We set $P_v = T_{ICP}P_p$. See Figure 3(d) for the result of this step.

ICP not only provides the pose P_v of K_v in the virtual system, but also the projection points $\bar{C_1}$ of C_1 onto the structure S_a. We therefore compute a connection between the physical point cloud and the virtual structure. In fact, for each tile t_i in S_a, we can determine how many projected points, called physical support points $s_i \subset \bar{C_1}$ of t_i are projected on t_i. The set of support points tells us, if a tile t_i of the virtual structure S_a is seen in the physical world. A tile t_i with a sufficient number of support points is present. However, the converse argument is not valid, since a tile without support could be physically present, but occluded. The next step, ray tracing, solves this problem.

3.4.2 Ray Tracing

This step determines the set of reflection points of a scan of the virtual Kinect K_v with pose P_v of the virtual building S_a. We position the virtual Kinect at pose P_v and simulate a ray-tracing using the Kinect's optical properties (resolution, view angles). Again, since the virtual building S_a is stored using planar polygons and a hierarchy of axis aligned bounding boxes, the ray intersection can be performed very efficiently. For each ray, we compute the closest intersection with a tile t_i from the Kinect, resulting in a virtual point cloud C_v. For each point in C_v, we know the supported tile t_i (i.e. the tile the generating ray intersected with). Ray tracing determines the support sets in the virtual system, that is, the support that we *should* see under the condition $S_a = S_p$. In contrast, $\bar{C_1}$ determines the real support, i.e. the support we *do* see. See Figure 3(e) for the result of the ray tracing step.

3.4.3 Tile Classification

Differences in support from C_v and $\bar{C_1}$ respectively determine if a tile is classified as present, missing, or occluded.

For every tile t_i, denote the number of physical and virtual support points by p_i and v_i respectively. Define r as the minimum ratio between p_i and v_i, $r = \min(\frac{v_i}{p_i}, \frac{p_i}{v_i})$,

t_r is a threshold value for this ratio, set to 0.7. For our purpose, it proved to be sufficient to only compare the number of support points of each tile, *i.e.*, we are not explicitly using any geometric differences. Support below a threshold of 100 points is set to 0.

The state of a tile t_i of S_a reflects its presence in the physical structure S_p. We determine this state as follows:

- $v_i = 0 \Rightarrow$ the tile is occluded.
- $v_i \neq 0$ and $p_i = 0 \Rightarrow$ the tile is missing.
- $v_i \neq 0$ and $p_i \neq 0$ and $r \leq t_r \Rightarrow$ the tile is missing. This case implicitly tests geometric differences.
- $v_i \neq 0$ and $p_i \neq 0$ and $r > t_r \Rightarrow$ the tile is present.

If the state of a tile t_i is "missing", the building robots have to adjust. "Present" signals, that $t_i \in S_a$ and $t_i \in S_p$ at the expected position, the building process can continue. If a tile is in state "occluded', the VI-robot has to re-scan the building from a different position before the building process can proceed. See Figure 3(f) for an example.

3.5 Online Error Correction

The VI-robot(s) is responsible for assigning the replacement of any missing tiles it discovers. It does this by managing an auction for each block that should have been placed but is absent. Each assembly robot sends a message to the VI-robot(s) after placing a tile. The VI-robot monitors these messages to maintain a state vector q, where q_i is 1 if the block has been placed and 0 otherwise. After each placement, the VI-robot reports a sensing vector q^s_j, where q^s_j is 1 if the block is definitely present, -1 if it is missing, and 0 if the presence or absence of the block cannot be determined. Then, if $q_i * q^s_j = -1$, a block that a robot claims to have placed is determined to be missing. Once the error has been detected, the scanning robot sends a message to inform the assembly robots that the block is missing and asks for bids to determine which robot will replace the missing block. Each robot then constructs a bid based on the following criteria:

$$b_i = w_i - A * c_i + B * d_{ij}, \tag{2}$$

where b_i is the bid of the i^{th} robot, w_i is the remaining workload of the i^{th} robot, c_i is the number of blocks still to be placed that are directly supported by the missing block, and d_{ij} is the distance between the missing block and the i^{th} robot's cache. The constants A and B are weights that can be optimized experimentally.

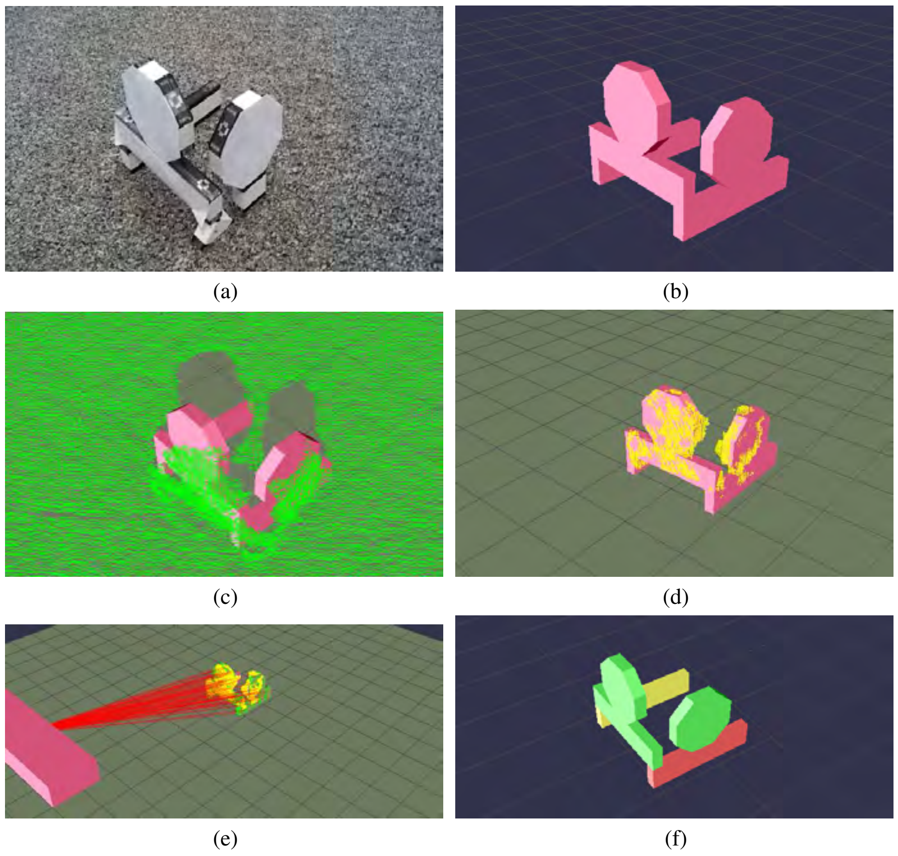


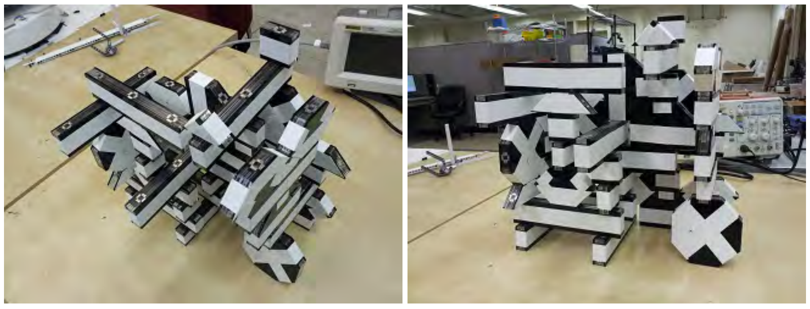
Fig. 3 Vision Feedback System. (a) Physical structure S_p (b) Virtual structure S_a. Note that S_p and S_a differ in this example: the long rectangle (front right) in the virtual structure is not present in S_p, it is replaced by a small cube. In the building process, this is an example of a missing/incorrect tile. (c) Green points show raw input data C_p from the physical Kinect sensor K_p. The Kinect's pose P_v in the virtual system was determined by the overhead positioning system. This figure shows the coordinate matching before floor based correction and ICP (d) After floor-based correction, ICP and candidate filtering: the yellow dots show the pose-corrected raw kinect data C_1, aligned to the virtual building. Points of the original raw data which were unlikely to support the structure were removed (floor- and bounding box based filtering). (e) Ray tracing: the red lines show some rays of the simulated Kinect K_v scan to determine the visibility of tiles $t_i \in S_a$. Yellow dots show the aligned real data C_1, green dots the result of the virtual scan C_v. The difference in support for each tile from yellow and green dots (real/virtual support points) is used to determine the state of each tile. (f) Result: Green tiles: present in S_a and S_v. Yellow tile: occluded (please note that this tile is occluded from view point P_v, as seen in (e). Here we rotated the view to make it visible). Red tile: Missing in S_p. The vision system correctly identified the front right rectangle as missing.

4 Simulation Results

We implemented the node trading algorithm on a network of seven computers communicating over a wireless router, each simulating the activities of a single robot.

Building a node was simulated by subjecting the robot to a delay randomly drawn from $\mathscr{N}(4,8)$, truncated at 0. We used a high standard deviation relative to the mean to test the system's robustness to high levels of variability. We analyzed two trends in the scaling of the problem. First, we considered all seven simulated robots cooperatively building a structure that varies in size from 27 up to 512 nodes. Second, we considered a variable number of robots applied to the same structure of 512 nodes. In both cases, we were primarily interested in how the completion time and the number of messages sent scale with the structure size and the number of robots.

The structures used for the experiments described above were cubes, built out of cubic pieces~~, as shown in Fig. **??**~~. The purpose of using a uniform structure is to isolate the effects of changes in structure size and number of robots. In table 1, we provide results of three specific experiments to demonstrate the flexibility of the approach. Example 11 corresponds to the structure shown in Fig. 4, which is one of the structures our experimental testbed can build, built here by 4 robots. This demonstrates the approach's ability to handle heterogeneous building materials in a variety of configurations. Example 12 and 13 are each based on a cube of 512 nodes built with 7 robots. In Example 12, we deliberately start one of the robots with only a single node to force the approach to recover from a lopsided workload distribution. Example 13 has the added constraint that each node with a y coordinate equal to 4 must be built before the adjacent node with a y coordinate equal to 5. This is to demonstrate the ability to manage types of constraints other than those imposed by gravity.

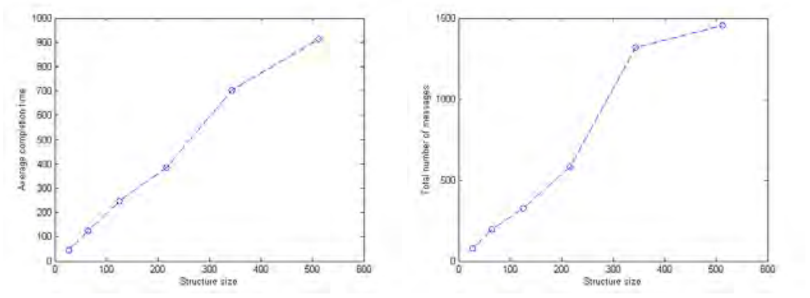


(a) (b)

Fig. 4 The structure used as example 11

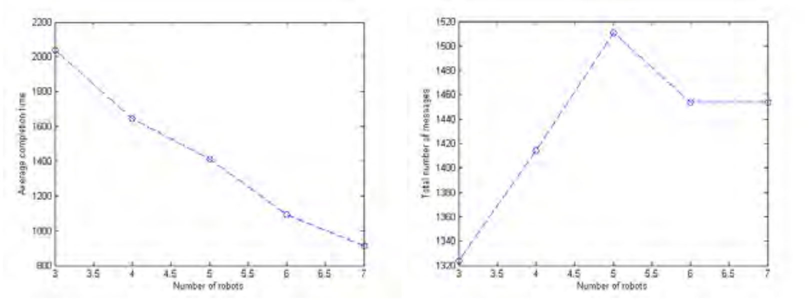
Table 1 Examples demonstrating flexibility

Criterion	Ex. 11	Ex. 12	Ex. 13
Structure size	43	512	512
# of robots	4	7	7
Ave. # of nodes per task	10.8	73.1	73.1
St. Dev. of # of nodes per task	3.30	35.4	37.6
Total # of nodes traded	15	55	15
Ave. wait time during construction	192.3	17.5	33.3
Ave. wait time after construction	11.8	236.4	775.1
Ave. completion time	302.7	1063.0	992.9
# of messages sent	508	1749	1615



(a) (b)

Fig. 5 Graphs showing relatively linear growth in both completion time and number of messages with respect to structure size. Data listed in table 5.



(a) (b)

Fig. 6 Graphs showing number of messages and completion time relative to number of robots. As the size of the team increases, the completion time drops linearly while the number of messages shows slight growth. Data listed in table 6.

5 Discussion of Simulation Results

Fig. 5 shows the approach being run with 7 robots on a variety of structure sizes. Examining the number of messages sent, we see that the communication volume increases linearly with the size of the structure. Average finish time is also increasing linearly with the size of the structure. This means that the rate at which the robots build is unaffected by the size of the structure. Also, it is worth noting that the amount of messages per time (number of messages divided by average completion time) is a constant relative to structure size, meaning there is no communication barrier to scaling this approach to larger structures.

Next, Fig. 6 shows how the approach scales with number of robots on a constant structure. First, the number of messages sent stays fairly constant across different numbers of robots, showing only a slight growth, so adding more robots does not cause any problem in communications. For the structure used, multiplying the number of robots used by the average completion time gives a relatively constant result. This means that adding an extra robot does not introduce significantly extra inefficiency in terms of idle robots waiting for placements. Certainly at some number of robots, this would stop being the case, but up through seven robots we have not yet seen significant additional idling. For this approach, a hard limit on the number of robots that can be used is the number of valid root nodes (recall that the root needs to be viable as the last node placed on the structure). For the structure used, this would allow up to 64 robots to be applied to the problem.

Example 12 started one robot with a single tile to test the robustness of the algorithm to lopsided initial conditions. As shown in Table 1, this resulted in a large number of nodes being traded,which is a result of nodes being moved towards the robot that started with a single node. This robot ended the experiment having built 25 nodes, roughly a third of the average size. This also resulted in an unusually high variance in number of nodes built during this experiment. Example 13 had a wall of constraints requiring that each node with a y coordinate of 4 be built before the corresponding node with a y coordinate of 5. This resulted in a bimodal distribution for number of robots built, with two robots building 128 nodes each and the other 5 all in the range 45-55. This is likely caused by an inability to trade across the wall of constraints imposed, which effectively divided the robots into two separate teams based on the locations of their roots.

6 Experimental Validation

6.1 Setup

The proposed distributed assembly strategy was implemented on our multi-robot assembly testbed. The testbed consists of two mini-mobile manipulators (M3 robots), or $N_c = 2$, shown in Figure 7, each equipped with an iRobot Create base, a Crustcrawler 5 DOF arm, 802.11b wireless communication, and a Hokuyo URG laser range finder (LRF). The laser range finder is first used to detect the position and orientation of the tile with respect to the robot. The arm is equipped with a 1 degree of freedom 2 finger gripper. The arm is then commanded to pick up or place the block at the given position. Tiles to be picked up are individually placed at fixed locations in the workspace. Since the geometry of each tile and the desired structure is known a priori, the desired placement position and orientation of each tile can be computed in relation to the base of the structure as detected by the laser range finder. Further, since each tile directly above another is defined as being constrained by the supporting tile, the robots assume the tiles above the current tile have not been placed yet, allowing us to assume an obstacle free path for the manipulator. Tiles attach and self-align using magnets. In addition to the two M3 robots, the testbed included one scanning robot equipped with a iRobot Create base and a Microsoft Kinect visual depth sensor. This robot follows a fixed rectangle surrounding the workspace of the assembly robots. Overhead localization for the robots was provided using two visual cameras.

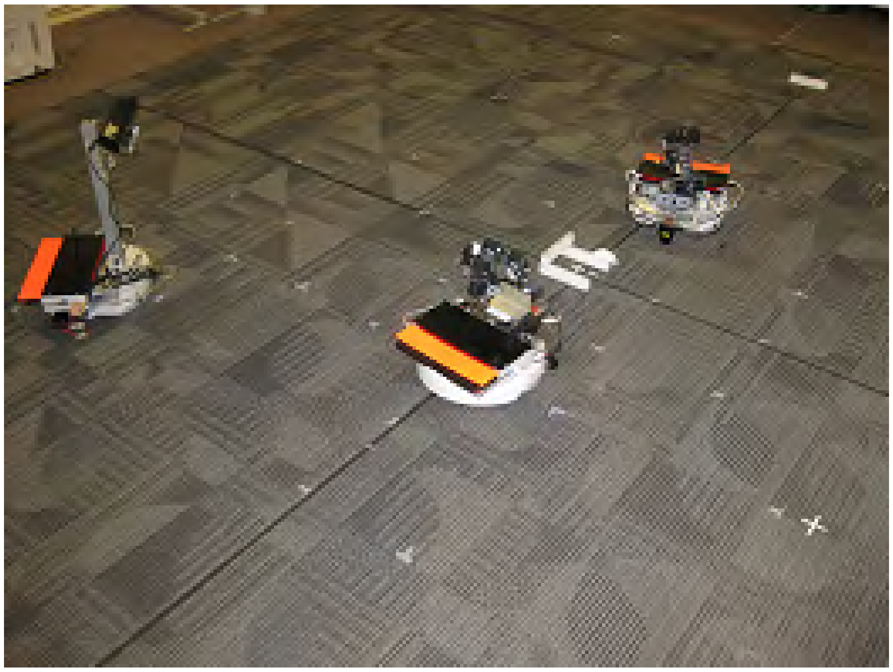


Fig. 7 Team of two assembly robots and one VI-robot with a raised Kinect around a partially completed structure.

Each robot was given the global position of the structure's center and the positions of their respective parts cache. The assembly parts were plastic tiles of various shapes and sizes (side lengths from $4-17$ *cm*), each with a given set of magnetic attachment sites (see Figure 8(a)).

To test error correction, each robot was assigned a preplanned assembly plan determined by Worcester et al (2011), with no node trading allowed. The assembly plans consisted of a list of tile identifiers in the computed assembly order. Distributed implementation of the plan was achieved by encoding the immediate supports for each component in the plan to ensure robots wait for the placement of a missing support tile by another robot before placing their parts.

The assembly tiles were grouped by type and placed in predefined locations in the workspace. The idea is to have a separate parts cache for each tile type. In our experiments, we considered the distributed assembly of 3D structures composed of 14 tiles with 5 distinct tile types. Figure 8(b) shows the desired structure for the experiment. To simulate missed placements, random assembly tiles were removed at various times during the assembly process.

In the next set of experiments the node trading was included to test for a reduction in assembly time compared to the preplanned approach. Tests were also done with one robot being given artificially longer assembly times to simulate the effects of a less efficient robot.

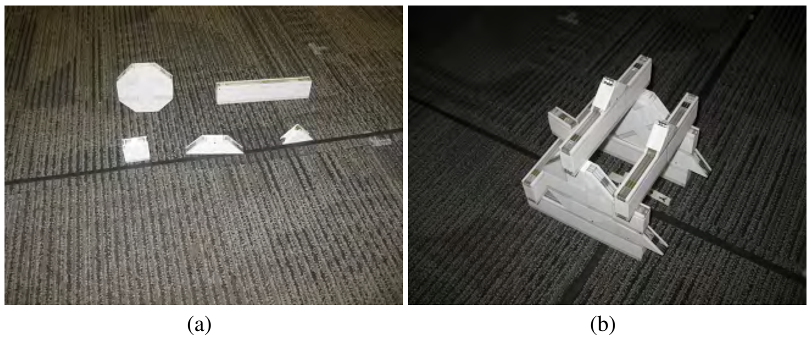


Fig. 8 (a) Sample assembly tiles. (b) Desired structure to be assembled.

6.2 Experimental Results

Fourteen experimental trials were run on the scanning robot for the desired structure shown in Figure 8(b). During each trial, one or more random assembly tiles were removed at different parts of the assembly process. Figure 9 shows the results of one of the experimental trials where the missing tile was successfully detected by the scanning robot. Out of twenty-two removed blocks, the scanning robot was able to successfully detect twelve of the missing tiles and reported undetermined for the other ten. There were no false positives during these trials, and only one false negative where a tile was reported as missing when it was actually present. The smaller tiles (square and triangle) were always reported as undetermined, while the larger tiles were always detected as missing after they had been removed in these trials.

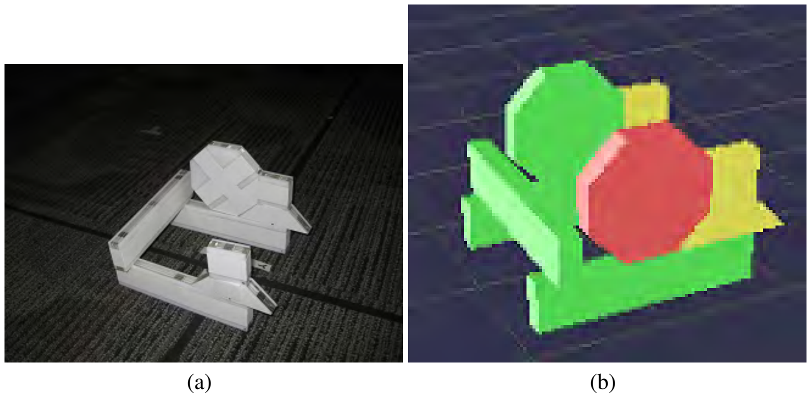


Fig. 9 ((a) Tile removed. (b) Missing tile reported by the scanning robot.

Table 1 summarizes the assembly partition obtained at the start of an experimental trial for each robot. The tiles allocated to each robot are shown in the order in which they are supposed to be placed. Table 2 shows the updated assembly allocation as tiles are removed during the experiment, including the workload reallocation after the detection of errors.

Robot 1	Tile ID	**Robot 2**	Tile ID
Long Rectangle	3	Trapezoid	7
Trapezoid	6	Octagon	5
Octagon	4	Square	8
Square	9	Square	10
Long Rectangle	11	Long Rectangle	12
Triangle	13	Triangle	14

Table 2 Initial Allocation for the 3D Structure in Fig. 8(b)

To test the node trading, experiments were done with varying simulated manipulation times (the amount of time taken to pick up or place a block), using two of the M3 robots described above. The manipulation was replaced with a timed delay drawn from a specified distribution. The typical manipulation times were measured to have a mean of 69.0 seconds and a variance of 2.5 seconds. The first set of experiments is done with this scaled down by a factor of 10. For the second set of experiments, robot 2 keeps the same distribution while robot 1's times are scaled up to half the original times (5 times that of robot 2). This is done in order to create a situation where the slower robot will need to give work to the faster robot in order to minimize completion time. Another benefit of running the experiments this way is that the unreliable placement does not need to be corrected with a scanning robot,

Robot 1	Tile ID	**Robot 2**	Tile ID
Long Rectangle	3	Trapezoid	7
		Removed tile	7
Trapezoid	6		
Octagon	4	Trapezoid	7
Removed tile	4		
Square	9	Octagon	5
		Removed tile	5
Octagon	4	Square	8
		Removed tile	8
		Octagon	5
Long Rectangle	11	Square	10
Triangle	13	Long Rectangle	12
Removed tile	13		
Square	8	Triangle	14
		Triangle	13

Table 3 Allocation After Detection of a Missing Tile.

allowing the node trading to be analysed on its own. Table 4 shows results for the two sets of experiments. From the results, it does not seem that the higher manipulation time was sufficient to cause any trades to occur for such a small structure. In fact, one of the trials with the higher manipulation time finished before the faster manipulation time. Since this was adding about 20 seconds per placement for 5 placements, it should have added 100 seconds to the completion time for the slower robot. However, the variation in time taken to navigate and use the LRF to align with the cache and structure seems to have been large enough to wash out the effect of a higher manipulation time. Larger experiments are needed to see the effects visible in the simulations.

Table 4 Two M3 robots with variable manipulation times

Criterion	Same speed	Different speed	Different speed, no trading
Structure Size	9	9	9
Ave. # of nodes	4.5	4.5	4.5
St. Dev. of # of nodes	0.71	0.71	0.71
Total # of nodes traded	0	0	0
Ave. wait time during construction	0.6	0.4	0.6
Ave. wait time after construction	165.8	220.4	156.3
Ave. completion time	1007.0	1066.4	953.0

7 Experimental Insights and Lessons Learned

The execution of complex tasks by a team of heterogeneous robots in a complex and dynamic environment with limited resources poses significant challenges. Most existing assembly strategies do not explicitly address the impact of sensing and actuation noise on the performance of a team of autonomous robots tasked to assemble complex three dimensional structures in an actual physical space. In our work, we consider the real-time on-board sensing requirements necessary for online adaptation of any distributed assembly strategy.

In our experimental setup, we considered two types of real-time on-board sensing: 1) the ability to localize the individual assembly tiles for pick-up and placement by the assembly robots, and 2) the ability to determine the state of the assembly structure during the entire assembly process. In both cases, the relative small size of the assembly tiles in relation to the sensing and actuation precision of the actuators and sensors used in the system posed significant engineering challenges. However, the ability to overcome these limitations at the small scale suggests that one can be more confident in the performance of the algorithms when employed on larger full scale systems.

8 Future Work

In this work, we presented a distributed 3D assembly strategy with online visual feedback to enable realtime error detection and correction. Our approach enables the online verification and adaptation of general 3D assembly strategies. An immediate direction for future work is to improve the visual feedback system to provide more detailed assessment of the state of the assembly structure. In particular, the reduction of false negatives by visually inspecting the structure via different viewpoints. A second direction for future work is to extend the visual feedback system to enable identification of incorrect assembly placements as well as missing tiles. The path of the scanning robot, which is currently a rectangle around the workspace, could also be optimized to ensure each node gets more frequent views. Expanding the size and reliability of the experimental testbed will allow more informative results to be seen. Finally, in both simulation and experiment it is worth noting that in several cases we do see significant idle time at the end of the experiment, meaning that robots that have completed their own tasks do sometimes have a long delay before the last robot finishes. This means that there is still room to gain from adding a mechanism for a robot to split its task in half, giving half to an idle robot. The complication with this will be revising the approach to allow two or more robots to share a root node, which will require all of them to confirm they are finished before one is allowed to build the root. Another way would be to split off a subtree with its own valid root node.

9 Acknowledgments

This work was supported by the National Science Foundation [CNS-1143941]; and the National Institute of Standards and Technology [ARRA-NIST-10D012].

10 Appendix A: Tables of Results

Tables 5 and 6 summarize the following quantities for each experiment: structure size, mean and standard deviation for number of nodes built per robot, total number of nodes traded, average time spent waiting during construction, average time waiting after construction (waiting for other robots to finish), average completion time (not including time spent waiting for other robots to finish, but including any wait time during construction), number of messages sent, and number of messages that were not acknowledged (giving an idea of how many messages are actually being received).

Table 5 Variable size cubes built with seven robots

Criterion	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5	Ex. 6
Structure Size	27	64	125	216	343	512
Ave. # of nodes	3.9	9.1	17.9	30.9	49.0	73.1
St. Dev. of # of nodes	0.38	2.19	0.690	1.57	3.00	2.19
# of nodes traded	0	3	0	3	23	11
Ave. wait time during construction	3.0	4.5	2.4	2.5	92.7	22.7
Ave. wait time after construction	22.8	25.6	51.7	37.2	55.0	33.7
Ave. completion time	44.0	123.7	246.8	383.6	704.3	912.6
# of messages sent	76	193	325	579	1316	1454

Table 6 Variable numbers of robots on a structure with 512 nodes

Criterion	Ex. 7	Ex. 8	Ex. 9	Ex. 10	Ex. 6
# of robots	3	4	5	6	7
Ave. # of nodes	170.7	128	102.4	85.3	73.1
St. Dev. of # of nodes	2.08	5.48	3.29	1.03	2.19
# of nodes traded	6	9	13	7	11
Ave. wait time during construction	1.1	3.9	60.8	38.1	22.7
Ave. wait time after construction	56.1	31.6	98.0	156.6	33.7
Ave. completion time	2038.2	1647	1413.7	1092.7	912.6
# of messages sent	1324	1414	1511	1454	1454

11 Appendix B: Index to Multimedia Extensions

The multimedia extension to this article is available on the IJRR website.

Extension	Type	Description
1	Video	Error correction by assembly and scanning robot

References

Besl P, McKay N (1992) A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 14:239–256

Chaimowicz L, Sugar T, Kumar V, Campos MFM (2001) An Architecture for Tightly Coupled Multi-Robot Cooperation. In: Proc. IEEE Int. Conf. on Rob. & Autom., Seoul, Korea, pp 2292–2297

Evans WC, Mermoud G, Martinoli A (2010) Comparing and modeling distributed control strategies for miniature self-assembling robots. In: in the Proc. of the 2010 Int. Conf. on Robotics and Automation (ICRA10), Anchorage, AK, pp 1438–1445

Fink J, Hsieh MA, Kumar V (2008) Multi-robot manipulation via caging in environments with obstacles. In: Proc. IEEE International Conference on Robotics and Automation (ICRA08), Pasadena, CA, pp 1471–1476

Grushin A, Reggia JA (2008) Automated design of distributed control rules for the self-assembly of prespecified artificial structures. Robotics and Autonomous Systems pp 334–359

Heger F, Singh S (2010) Robust robotic assembly through contingencies, plan repair and re-planning. In: Proceedings of ICRA 2010

Klavins E (2007) Programmable Self-Assembly. In: Control Systems Magazine, vol 24, pp 43–56

Lindsey Q, Kumar V (2012) Distributed construction of truss structures. In: Proc. of Workshop on the Algorithmic Foundations of Robotics (WAFR)

Mataric MJ, Nilsson M, Simsarian K (1995) Cooperative Multi-Robot Box-Pushing. In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS95), Pittsburgh, Pennsylvania, pp 556–561

Matthey L, Berman S, Kumar V (2009) Stochastic Strategies for a Swarm Robotic Assembly System. In: Proc. 2009 IEEE International Conference on Robotics and Automation (ICRA09), Kobe, Japan, pp 1953–1958

Nüchter A, Lingemann K, Hertzberg J, Surmann H (2005) Heuristics-Based Laser Scan Matching for Outdoor 6D Slam. In: KI 2005: Advances in Artificial Intelligence. 28th Annual German Conference on AI, Proceedings Springer LNAI vol. 3698, pages 304 - 319, Koblenz, Germany

Petersen K, Nagpal R, Werfel J (2011) Termes: an autonomous robotic system for three-dimensional collective construction. In: Robotics: Science and Systems VII

Rai V, van Rossum A, Correll N (2011) Self-assembly of modular robots from finite number of modules using graph grammars. In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS11), San Francisco, CA

Sanderson AC, Zhang H, Mello LSHD, S AAC, S AC, Zhang H, Zhang H, Mello LSHD, Mello LSHD (1990) Assembly sequence planning. AI Magazine 11:62–81

Stein D, Schoen R, Rus D (2011) Constraint-aware coordinated construction of generic structures. In: IEEE International Conference on Intelligent Robots and Systems (IROS11), San Francisco, CA

Werfel J, Nagpal R (2008) Three-dimensional construction with mobile robots and modular blocks. In: International Journal of Robotics Research, vol 27, pp 463–479

Worcester J, Rogoff J, Hsieh MA (2011) Constrained task partitioning for distributed assembly. Proc 2011 Int Conf on Intelligent Robots and Systems (IROS11)

Yun SK, Rus D (2010) Adaptation to robot failures and shape change in decentralized construction. In: Proc. of the Int. Conf. on Robotics & Automation (ICRA10), Anchorage, AK USA, pp 2451 – 2458

Yun SK, Schwager M, Rus D (2009) Coordinating construction of truss structures using distributed equal-mass partitioning. In: Proc. of the 14th International Symposium on Robotics Research, Lucerne, Switzerland