Coordinating Construction by a Distributed Multi-robot System

by

Seung-kook Yun

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Electrical Engineering and Computer Science
Jul 13, 2010

Certified by .....
Daniela Rus
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Coordinating Construction by

a Distributed Multi-robot System

by

Seung-kook Yun

Submitted to the Department of Electrical Engineering and Computer Science
on Jul 13, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis presents a decentralized algorithm for the coordinated assembly of 3D objects that consist of multiple types of parts, using a networked team of robots. We describe the algorithm and analyze its stability and adaptation properties. We partition construction in two tasks, tool delivery and assembly. Each task is performed by a networked team of specialized robots. We analyze the performance of the algorithms using the balls into bins problem, and show their adaptation to failure of robots, dynamic constraints, multiple types of elements and reconfiguration. We instantiate the algorithm to building truss-like objects using rods and connectors. The algorithm has been implmented in simulation and results for constructing 2D and 3D parts are shown. Finally, we describe hardware implementation of the algorithms where mobile manipulators assemble smarts parts with IR beacons.

Thesis Supervisor: Daniela Rus
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I am deeply grateful to my advisor, Daniela Rus. She has been an excellent mentor and a great guider for my research.

I would like to thank to the Distributed Robotics Laboratory members for their help in every stage of this work. Marsette Vona and Carrick Detweiler have helped me from the first stage of my lab life, and Mac Schwarger has been a great colleague as well as a next-door friend. This thesis could not be done without the young guns - Adrienne Bolger, Matt Faulkner, David Stein and Lauren White - who shared the bitter failures as well as the sweet successes. I wish their best in the future and hope them to continue experiencing the fun world of robotics, after all their efforts with me. I thank Sejoon Lim and Byungkwon Ahn for their friendship in the lab.

I am also grateful for the Korean EECS members at MIT.

Most of all, I would like to express my deepest love for my family. My wife - Jihyun Kim - has been the half of myself. I admit I could have done nothing without her. My beloved sons - Yeomin Yun and Yeojun Yun - always make me encouraged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are interested in automating assembly and construction tasks. Currently assembly, whether for complex places or buildings, is done using manually generated blue prints and many skilled human workers. The completion time is dependent on the human performance and regulated with respect to the number of hours a human team can perform. In this thesis, we explore the use of robots to automate and optimize assembly tasks. Robots can perform tasks that are (1) hard for humans (e.g. lifting parts), (2) dangerous for humans (e.g. building a tall scaffold) and (3) they can be more efficient and accurate than humans. Assembly line robotic automation has the further benefit of relieving humans from having to execute tasks for long periods of time in ergonomically difficult positions.

We see construction as an important application for robotics because the nature of construction is complex, yet it includes many routine jobs in which the robots have to pick up and place regularly shaped source parts such as trusses, blocks, windows, and etc. The state of the art in robotics for construction and assembly has yet to come to the versatility of a human; however robots have proven to be more effective than humans for limited scope operations including handling parts in a structured environment (for example, assembly lines in factories). Many algorithms used in assembly can be applied directly or indirectly to construction: task assignment algorithm, parallel algorithm, manipulation, navigation, and many more.

Robotic construction is challenging because automating manipulation operations is hard. One option for automation is to have robots learn from humans and copy their actions

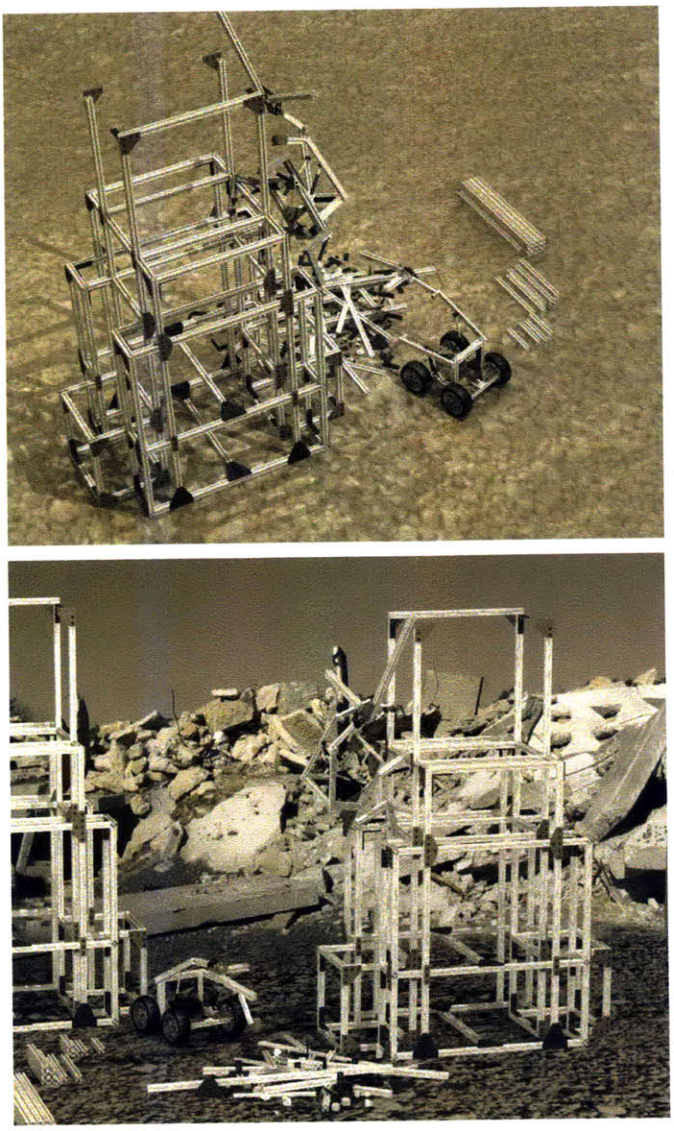


Figure 1-1: Concept art for construction of a truss structure by mobile delivering robots and truss-climbing assembling robots. Reprinted with permission from Jonathan Hiller, Cornell University, USA.

by following them. This could enable the step-by-step execution of a blue print.

Step-by-step execution can be carried out in sequence by one or more robots; however effectively this method does not maximize the use of the robots with respect to parallelism, nor does it guarantee the fastest completion time of a task. In this thesis we develop new algorithms and systems that enable groups of robots to complete a complicated assembly task using the maximal amount of parallelism afforded by the team and the task. To see the difference between the benefits of such a method over incremental construction following a blueprint step-by-step, consider the example in Figure 1-3, where that task is to build a densely tesselated (grid-shaped) A-shaped structure using rods and connectors. Incremental construction by the robot team is one solution and the idea is illustrated in Figure 1-4. Note that the incremental construction requires that each robot travels the entire blueprint and this has several disadvantages. First, the robots move too much. Second, this hinders parallelism, thus slowing completion. Third, the centralized solution requires knowledge of the exact construction plan and the placement of each component ahead of execution time. This renders the algorithms unadaptive to the amount of source material. For the A-shape structure, we would like the flexibility to construct the scaffold as grid as dense as possible given the amount of source materials. In other words, the shape will be approximated as a coarse grid when the material is sparse, or as a dense grid when the material is abundant. Fourth, collision avoidance may be a larger challenge than necessary if the robot team is large and the robots attempt to work in the same cell.

We investigate a new approach to multi-robot construction that avoids the challenges of the incremental approach and leads to highly parallel assembly solutions and optimal construction times. Our vision for robotic assembly is illustrated in Figure 1, which shows a heterogeneous team of robots creating truss-like structures by cooperation to support parallelism and some specialization to enable efficiency in the performance of different tasks. Our model includes two types of robots: part delivery robots, specialized for locating and delivering parts, and assembly robots, specialized for joining the parts delivered to them into desired objects. We wish to develop decentralized algorithms for such heterogeneous teams that are (1) fully decentralized and distributed on the group, (2) adaptive to changes in the environment and the group, (3) provably convergent, and (4) experimentally feasible.

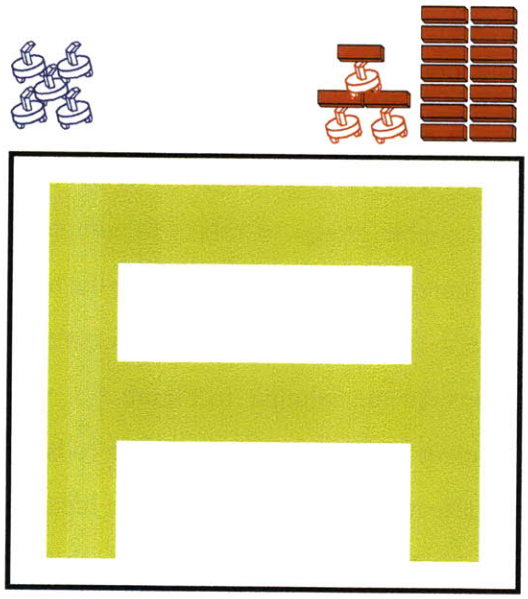


Figure 1-2: An example of building an a-shaped bridge. The yellow region is a blue print to be filled with red trusses stacked at the upper right corner. The blue robots are assembling robots and the red robots are delivery robots.

The distributed controller is desired for a large system because of its scalability; the same controller will work for thousand robots as well as a single robot. Adaptivity will make the controller flexible to change which can frequently happen during construction. Feasibility is essential for transition of a robot system from a lab to the real world.

A typical assembly scenario requires that parts of different types get delivered at the location where they are needed and incorporated into the structure to be assembled. We abstract this process with two operations: (1) tool and part delivery carried out by delivering robots, and (2) assembly carried out by assembling robots. In this thesis, we consider how a team of robots will coordinate to achieve assembling the desired object. Tool and part delivery requires robots capable of accurate navigation between the part cache and the assembly location. Assembly requires robots capable of complex grasping and manipulation operations, perhaps using tools. Different assembling robots work in parallel on different subcomponents of the desired object. The delivering robots deliver parts (of different types) in parallel, according to the sequence in which they are needed at the different assembling stations. For practical considerations, we consider the case where the parts are

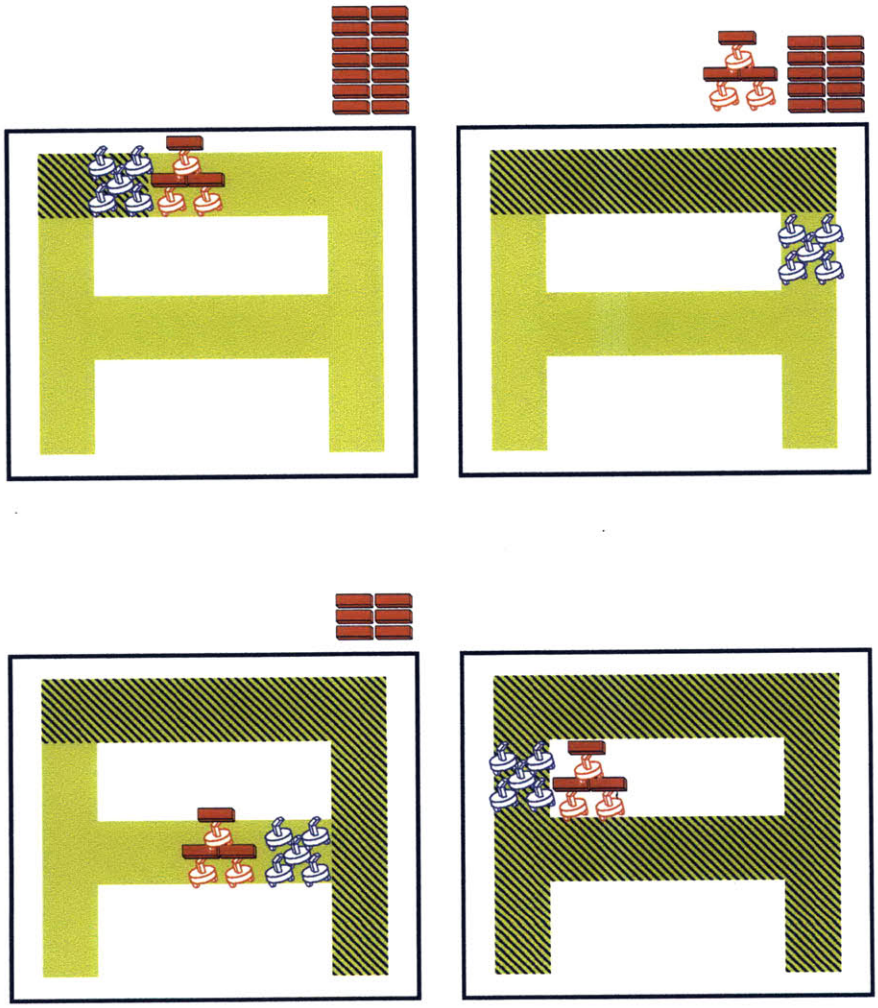


Figure 1-3: Snapshots of the incremental solution for robotic construction. A team of the assembly robots move together to build the structure block by block while the delivery robots keep carrying the parts from the source cache.

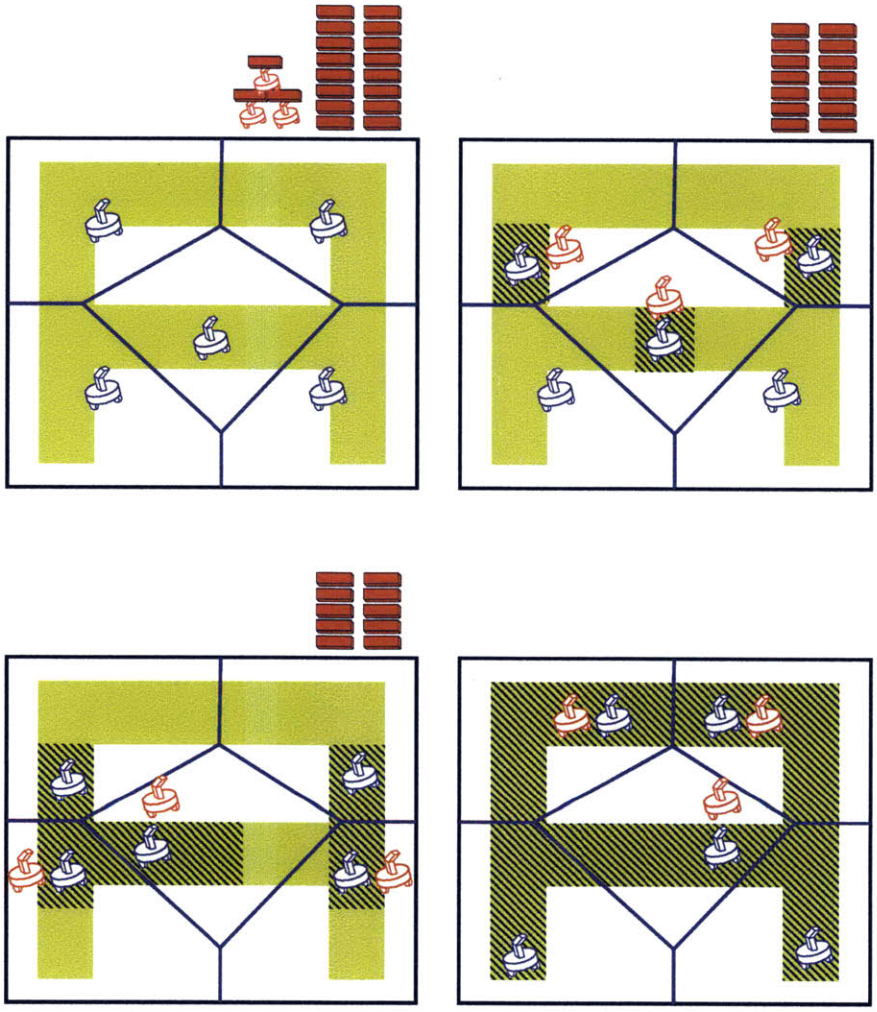


Figure 1-4: Snapshots of our solution for robotic construction. Each assembly robot now takes care of its own region decided by the Voronoi tessellation.

(a) rods of different lengths and (b) connectors for connecting the rods into truss-structured objects in our examples; however the algorithms are general and work with any parts. The robots can communicate locally to neighbors. The delivering robots have the ability to find the correct part type in the part cache, pick it up, and deliver it to the correct spot for the assembling process requesting the part, and return to the part cache for the next round of deliveries. The assembling robots have the ability to receive the part from a delivering robot and incorporate it into the assembly.

We assume that the target object is given by a material-density function which encodes the object geometry as a blue print and is known to all the robots. The construction process starts by a *coverage*-like process during which the assembling robots partition the target structure adaptively into sub-assemblies, such that each robot[1] is responsible for the completion of that section. To achieve this division, the robots locally compute a Voronoi partition, weighted by the mass of all the rods contained in the partition, and perform a gradient descent algorithm to balance the mass of the regions. We extend this algorithm to a discrete space where robots are in a graph composed of nodes with weights and edges. The node weight corresponds to a density in the continuous domain. We explore the difference from a continuous domain such as graph Voronoi tessellation and geodesic distance, and propose the decentralized controller which turns out to require two-hop communication rather than the one-hop communication required by the solution modeled using continuous space.

The delivery robots also know the density function. They locate parts in a part cache and bring the parts to the assembly robots. We wish to control this process so that each assembly robot proceeds with its task at approximately the same pace as the other robots. Since the overall assembly was partitioned into approximately equal parts, this process guarantees that all assembly cells complete at approximately the same time. We developed a new algorithm based on this intuition. Delivery occurs according to the demanding mass for each subassembly, that is, the amount of work that remains to be done, measured in the number of components that have yet to be added to the assembly. This idea can be

[1]The robot represents all the skills needed for each required assembly step; in some cases multiple robots will be needed, for example the connection of two rods with a screw is done by three robots, one robot holding each rod, and one robot placing the connector.

implemented as local search and guarantees global and local balance for part delivery.

A nice feature of our controllers for assembly and delivery is that the algorithms are adaptive in multiple ways. Robustness and adaptation are desirable in complex systems consisting of large numbers of robots, where failures and changes in the system flow may be expected. We show the control algorithms are (1) robust to a failure of robots, (2) adaptable to any order of construction, (3) capable of being used for reconfiguration between different truss structures, and (4) adaptive to human changes.

We have implemented the algorithms for part delivery and assembly using a hardware platform we designed and built in our lab. We used a team of 4 robots, two delivery robots and two assembling robots. We used instrumented parts (see Section 8) to simplify object location, grasping, and handoff. Each robot has an iCreate mobile base, a CrustCrawler 4DOF arm, and networked communication provided via the Meraki networking platform. Our experiments show that the algorithms are effective at executing the assembly and delivery tasks.

1.1 Decentralized Control Algorithms

More specifically, this thesis proposes decentralized control algorithms for partitioning, part delivery, and assembling steps. The partitioning algorithms are inspired by *distributed coverage* introduced in [23, 100, 85] and use *equal-mass partitioning* as the optimization criterion. The algorithms rely only on local information (e.g. neighbors exchange information about their local mass). The partitioning controller has a form of gradient descent with a cost function that is minimized when all the assembly robots are allocated with the same amount of subassemblies (partition).

Using the delivery algorithm, each delivery robot chooses an assembly robot by performing two steps: probabilistic deployment and local search for larger demanding mass. Probabilistic deployment to an assembly substation for the next component does not guarantee that the group of robots will finish the subassemblies at approximately the same time. Probabilistic deployment appears to guarantee a certain amount of the global balance, however, our analysis shows the balance breaks out as a number of delivery components in-

creases. The local search algorithm is introduced to augment the probabilistic deployment to ensure global balance in assembly completion. Local search works as follows. Each delivery robot arrives at a subassembly station according to the probabilistic deployment algorithm. At this point, the robot communicates with all the assembly robot neighbors to find out who has the greatest need for a part (that is, who has completed the least amount work). The delivery robot moves to that assembly station and repeats the query process until the assembly robot with the greatest need for the part is identified. We prove this local search very effective in reducing the unbalance.

In addition, the task allocation and part delivery algorithms are provably stable. They are adaptive to the number of delivering robots and assembling robots as well as to the amount of source material. We implemented these algorithms in simulation. Several 2D and 3D truss-structures were created using our algorithms. Our hardware implementation using 4 mobile manipulators shows that the algorithms are effective

1.2 Adaptive Construction

We extend the work to construction beyond trusses where the target consists of any given number of parts from a set of part types. The density function is modeled as a sum of separate density functions for each part, only if there is no dependency between the parts. The dependency will be left for the future work.

Also we show how the algorithms can be adapted to (1) the failure of changing numbers of assembly robots and delivery robots, (2) dynamic constraints such as order of construction, and (3) changes in the geometry of the target structure during assembly. We demonstrate the performance of the algorithm in simulation.

1.3 Extension to Discrete Space

Most of the distributed coverage algorithms operate in continuous domains and controllers for distributed coverage require a convex target area in Euclidean space as well as a continuous weighting function on the target area. However, many applications in assembly,

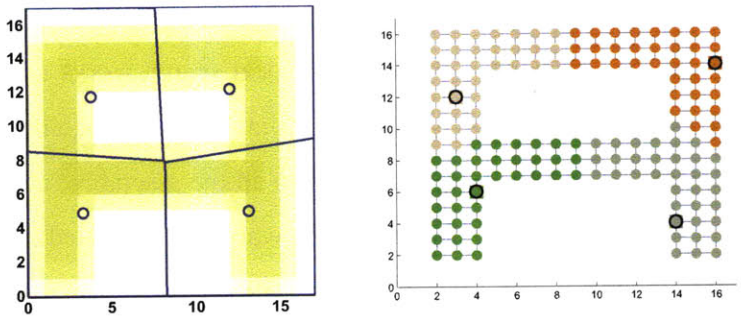


Figure 1-5: Simulation result from 4 robot coverage on an A-shaped bridge. The final configuration of equal-mass partitioning (a) in a continuous domain and (b) on a graph.

construction, transportation and resource allocation require coverage to operate in discrete domains. For example, coverage of a structure that can be explicitly modeled as a graph, such as a truss structure where truss elements are modeled as edges and connectors/screws are represented as nodes [65]. Distributed coverage in a non-convex region with obstacles is also possible with a mesh network in the target region.

In this thesis we describe two decentralized algorithms for distributed coverage on a graph. The algorithms use a geometric approach based on graph Voronoi tessellation which converges when robots reach the weighted Voronoi centroids. A node-weighting function represents the importance of nodes. We investigate two types of coverage: (1) locational optimization where a team of robots looks for the optimal set of locations with respect to node weights, and (2) equal-mass partitioning which distributes equal weights to each robot, and show that these problems are related and solvable with similar techniques. The algorithms use vertex substitution to sequentially find the best partitions by checking every possible movement of a single centroid (robot position in our case). We prove convergence of the algorithms to local minima in solution space and experimentally demonstrate that a large fraction of the solutions found by our algorithms are statistically close to the global optima. A surprising result is that two-hop communication rather than single hop communication to neighbors is required for the best performance.

We show results from an implementation of the algorithms on two graph topologies which represent blue prints of bridges.

1.4 Experiment for Distributed Robotic Construction

We discuss the differences between the theoretical and the practical algorithms and present data from extensive subassembly partitioning and tool delivery experiments. We also discuss data from a preliminary planar implementation of the assembly algorithm that places the parts in the correct sequence. We have implemented the decentralized construction algorithms on a platform with 4 robots. The task is to build a planar truss. The system takes as input the specifications of an object to be assembled from rods and connectors, causes the robots (1) to identify the subassemblies that can be created in parallel, (2) deliver parts to each subassembly team so that the subassemblies are created in approximately the same amount of time, and (3) place the parts in the required sequence to construct the desired object. We use smart parts for the assembly. The smart parts have embedded two-way communication systems that allow the parts to transmit their location (in the form of a beacon) as well as their geometric and mass properties to the robots. The robots use communication-enhanced grippers to locate, identify and grasp the objects. Our solutions to problems (1) and (2) are general with respect to this grasping modality. Our solution to problem (3) applies to planar objects and illustrates the correct position of the parts. The actual assembly to create a rigid object is not yet solved.

The robot system for construction is composed of 4 mobile robots with a 4-dof manipulator and two kinds of components (truss and connector) with embedded IR beacon for communication with the robots. Each robot is also equipped with communication devices for localization, inter-robot communication, and robot-part communication. The theoretical algorithms in [65, 103] guarantee stable and convergent controllers, but moving from theory to hardware implementation requires changing the original assumptions and the algorithmic details that rely on them.

1.5 Summary of Contribution

The main contributions of this thesis are:

1. proposal of distributed processes for coordinated robotic construction (Algorithm 1),

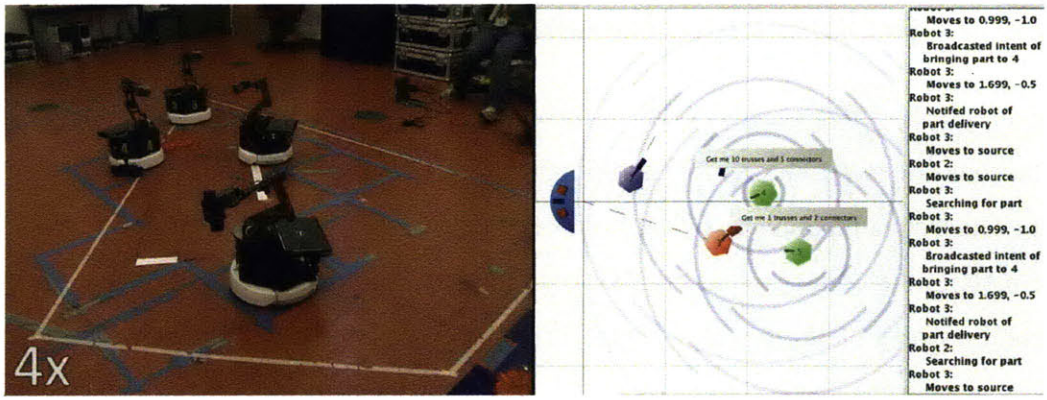


Figure 1-6: 4 mobile manipulators are delivering and assembling smart parts with IR beacons. A snapshot of the GUI is on the right.

2. the distributed algorithm for equal mass partitioning of an assembly task using a continuous space formulation (Section 4.2),

3. the provably correct decentralized delivery and assembly algorithms (Algorithm 5 and 6),

4. the two distributed algorithms for coverage on a graph (Algorithm 3 and 4),

5. the communication condition required for convergence of the distributed partitioning algorithms on a graph (Section 5.2.1),

6. the convergence proofs and evaluation for the algorithms (Section 4.2.1),

7. the development of the hardware platform with the mobile manipulators,

8. the experimental implementation of the algorithms with the smart parts (Chapter 8).

1.6 Organization

This thesis is organized as following. In Chapter 2, we survey previous work in robotic construction as well as distributed algorithms for coverage. We formulate our problem in Chapter 3. The decentralized partitioning algorithms for assembly assignment are introduced in Chapter 4 and Chapter 5 for a continuous domain and a discrete domain, respec-

tively. Chapter 6 shows the control algorithms for delivery and assembly in a distributed way and analyze the algorithms based on the *balls into bins* problem. The flexibility and adaptability of the proposed algorithms are discussed in Chapter 7. In Chapter 8, the algorithms are implemented in the robotic system with mobile manipulators and smart parts.

Chapter 2

Related Work

The idea of robotic construction with a team of networked robots using elements from the environment is not new, for example see Matthey et al's paper on stochastic strategies for a swarm robotic construction [76] and references therein.

In this thesis, we explore the particular idea of maximizing the parallelism in construction by partitioning algorithms and efficient delivery algorithms, with an emphasis on stable and provably correct controllers.

Our proposed systems and algorithms are further related to prior work in the fields of robotic construction, distributed coverage, graph partitioning, and truss-handling robots.

2.1 Robotic Construction

Automated assembly is one of the most successful application of robotics. Factory-based robotic automation for assembly operation has been a great success for robotics. Fanuc, for example, has more robots than humans working in their plants. These robots operate in very fixed and structured environments and perform repetitive and simple tasks. Several important research projects have addressed how to extend this current use of robots to increasingly less-structured environments and increasingly more complex tasks. Construction can be viewed as a kind of assembly, often carried out outdoors. Construction and assembly share many of the challenges, properties, and desiderata. Next, we summarize some of the key recent results in this space.

In [34], Fitcher introduced a basic theory and practical implementation of a stewart platform based manipulator for construction.

Fahlman proposed a planning algorithm for robotic construction tasks [31], discussing dependency of the parts and their correct order. The algorithm generates a plan for constructing specified structures given simple block-like objects. The paper considered usage of extra blocks as temporary supports or counterweights during construction.

Nechba et al [82, 77, 78] launched a self-mobile space manipulator project, and they developed several robots including a truss-walking inspection robot SM^2 that manipulated and assembled space station trusses. The focus of the project was to design a tele-operated mobile robot with the controller for mobility and manipulability including gravity compensation.

Stroupe et al [106] built a space robot team which was able to demonstrate component placement to an already existing structure.

Staritz et al built a space robot Skyworker [105] which demonstrated truss-like assembly tasks. The robot is a manipulator with serially connected links, and its hardware design and workspace were reviewed for assembly tasks.

Werfel et al. [112, 109, 111, 97, 110] introduced a 3D construction algorithm for modular blocks. They assumed the blocks were capable of communicating to the robots, and the algorithm outputs a provably correct sequence of assembly without a deadlock. They compared performance in simulations by the robots with different capability.

Matthey et al's paper on stochastic strategies for a swarm robotic construction [76] simulated a robotic assembly by modeling change of robot states as chemical reaction equations.

Nease et al [81] introduced the approach of the Air Force for construction robotic technology which requires place construction and repair equipment.

Parker et al [84] implemented swarm construction algorithms on a small team of mobile robots for blind bulldozing so that the robots can prepare for space missions. Their experiments showed how a team of four robots prepare for *nests* by blind bulldozing for a couple of hours.

Lee et al [69, 21] built a robot manipulator equipped with pneumatic actuators for con-

struction. The robot was controlled half-remotely and half-autonomously.

Yamada et al [116, 117, 115, 119, 120] studied a tele-robotic system for construction with virtual reality. They made a servo-controlled construction robot which was tele-operated by two joysticks on a 3-dof mobile base. 3D computer graphic of a virtual robot as well as 3D stereo image were given to an operator to help efficient construction.

2.1.1 Modular robots for construction

Schweikardt built roBlocks [102], a robotic construction kit with cube-like modular robots that can interact to each other physically and electronically. There are many similar modular robots for construction, more specifically *self-assembly*, which normally have a form of lattice or chain.

Murata, et al's built "3D Fracta" [96] which works like a reconfigurable lattice. The robot unit has rotatable connectors on each side of a cube so that it can move another unit. A stochastic algorithm is used to control the units in a distributed way.

Kotay and Rus developed "Molecule"[57, 56, 55] which has male and female connectors to assemble it to another molecule and can lift up the connected molecule in 3D. The proposed controllers move a group of the molecules in a distributed fashion.

Rus and Vona built "Crystal"[25, 26, 27, 24] which expands and shrinks its body for 2D reconfiguration. They introduced an algorithm to move a cube from one location to another in a distributed way.

Unsal, Kiliccote, and Khosla made bi-partite "I-Cubes" [20] system which is heterogeneous with a cubic module and a link module. Centralized locomotion algorithms were used with given combinations of the modules.

Lund, Beck, Dalgaard, Støy et al developed ATRON [48, 53] which is a sphere rather than a lattice. Each unit has an upper and lower hemisphere and the structure lead to a complicated controller for 3D reconfiguration.

Duff, Yim, et al's PolyBot [28] is a chain-type module, and linked modules can reconfigure themselves to an arbitrary 3D chain. They showed how tens of the modules are coordinated to change a global structure such as from a four-legged robot to a snake or a

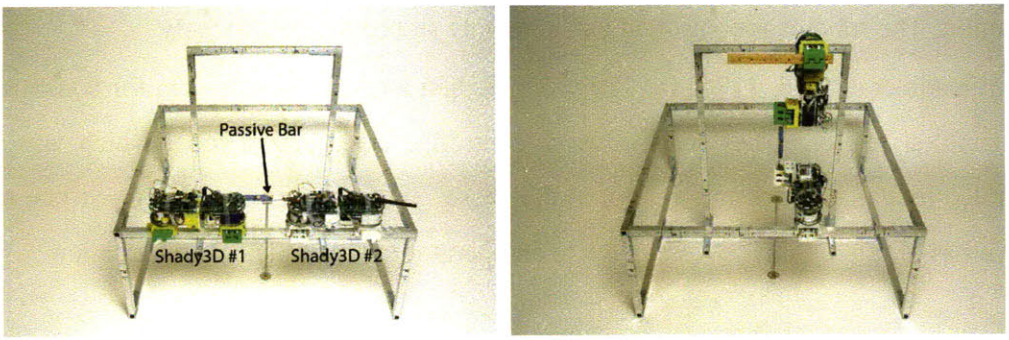


Figure 2-1: Implementation of pick-and-drop of a bar with two Shady3D. Reprinted from [19].

fully connected chain.

2.1.2 Our prior work for robotic construction

Our previous work on truss assembling robots includes Shady3D [63, 64, 19, 62, 66] that utilizes a passive bar with active communication and may include itself in a truss structure, and is controlled by locally optimized algorithms (See Figure 2-1).

We also proposed a centralized optimal algorithm to reconfigure a given truss structure to a target structure [61]. The concept is shown in Figure 2-2. The paper proposed an optimal set of paths for a robot to follow in terms of the total moving distance of trusses. The optimal set guarantees connectivity of the structure during reconfiguration.

This thesis introduces a new approach in which robots are specialized as delivery and assembly robots, and distributed algorithms control the assembly of a structure with multiple kinds of source materials.

2.2 Distributed Coverage of Multi-robot Systems

Our assignment algorithm for sub-assemblies is inspired by *distributed coverage* for a multi-robot system, which has been heavily studied to optimize locations of robots [23, 100], to find the best partition for vehicle routing [85], and to distribute workload equally [65].

We follow the notion of locational optimization developed by Cortes et al. [23], who introduced distributed coverage with mobile robots.

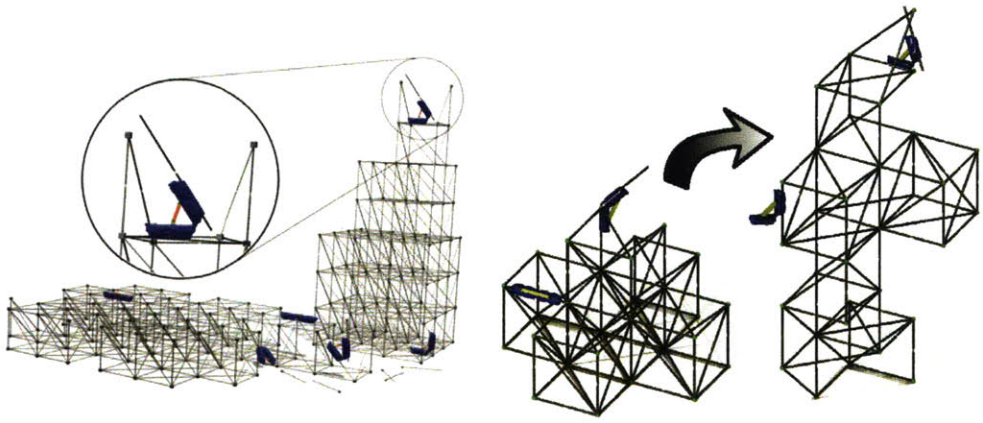


Figure 2-2: Artist rendition of several hinge robots decomposing and recomposing truss structures. Structural metabolism replicates properties of biological metabolism such as autonomous disassembly and assembly, continuous reuse of modular elements, automated design from functional requirements, and resilience to raw material variation. Reprinted from [61].

The same optimization criteria was used in a distributed coverage controller for real-time tracking by Pimenta et al. [87] in which a team of robots track a moving target while maintaining the optimal coverage configuration. The target was modeled as a time-varying factor in a density function so that it could be melt into in a framework of distributed coverage.

Schwager [100] et al used adaptive coverage control in which networked robots learn a sensory function while they are controlled for the locational optimization. The paper showed consensus among only neighboring robots dramatically improved the speed of convergence to the optimal configuration.

They also developed coverage algorithms to optimize camera placement for hovering agents in three dimension [98]. The cost function incorporated camera specs such as a focal length, and a gradient descent controller produced the optimal 3-D configuration of quad-rotor helicopters.

This thesis inherits the distributed coverage concept, and pursues *equal-mass partitioning* in which every networked robot is controlled to have the same amount of construction (in our case, truss elements and connectors) to be built, rather than optimal sensing locations.

Pavone et al. [85] have been independently working on equitable partitioning by the power diagram, which was designed for equal work load for vehicles in a vehicle routing problem.

Recently, the coverage algorithms have been extended to coverage of a non-convex region.

The visibility based deployment problem was tackled in [39], where a team of robots solve the art-gallery problem in which occlusion led to non-convexity.

In [18] a non-convex region is transformed to a convex region by a diffeomorphism.

[88] uses the geodesic distance measure for a non-convex region instead of Euclidean distance.

Controlling mobile robots with proximity constraints was addressed for a known environment with obstacles in [7].

The solutions work for specific environments, however finding a solution for all types of non-convex environment is still an open problem.

2.3 Graph Partitioning

Graph coverage and partitioning have been extensively studied in order to find the optimal locations of resources [107] and to distribute workload equally. For excellent surveys see [91, 36].

This thesis revisits two classic problems: the p-median problem and graph partitioning, which are NP-hard even in a centralized view.

The p-median problem is to find centroids of a graph which minimize the maximum distance between nodes and the centroids. Polynomial time algorithms only exist for a tree [42, 52], and all we can use for a general graph are heuristics such as greedy [68, 47], approximation algorithms [67, 104], alternate [75] and vertex substitution [3, 107]. Since the p-median problem can be re-written as an integer programming, LP relaxation [6, 11, 13] and the branch-and-bound heuristic [5, 4, 72] are widely used as an approximated solution.

The graph partitioning problem is to find subsets of a graph so that each subset has the

same total node weights while minimizing the cut size that is defined as the sum of weights of edges crossing between the subsets. Again the NP-hardness of the problem led to many heuristics. Sequential algorithms such as the KL method [59, 35], simulated annealing [60, 22, 114], tabu search [43, 44, 10], and genetic algorithm [45, 95, 74], locally improve partitions whereas global methods recursively bisect the graph until we have the wanted number of subsets, using the recursive coordinate bisection [15], the inertial method [32, 70, 79], the recursive spectral bisection method [89, 50, 8], etc.

These methods were used in robotics [38]. Using an environment discretized by grid cells, a centralized algorithm based on spanning trees directs the robots to cover the environment.

In [29] a group of mobile robots are deployed to cover a discretized environment by gossip communication. The paper proved convergence even with only one-to-one communication, though the amount of communication should be large.

We model a non-convex environment as a graph in which node weights correspond to density in a continuous domain, and propose decentralized graph partitioning algorithms for locational optimization and equal mass partitioning.

2.4 Variable Geometry Truss and Truss Climbing Robots

Variable geometry trusses (VGTs) can be viewed as a generalization of the serial-chain hyper-redundant systems to more general kinematic topologies. Both fixed-topology systems like the NASA/DOE "SERS DM" [93] and manually-reconfigurable systems—notably Hamlin, Sanderson, et al's TETROBOT [46]—have been considered. Also related are robotic systems which assemble static trusses, for example, Everest, Shen, et al's SOLAR [51], and Howe and Gibson's "Trigon" system [1]. Such self-assembling and self-reconfiguring truss systems are a promising direction for robotic assembly of large structures in space—for example, see Doggett's overview of automatic structural assembly for NASA [113].

Truss *climbing* robots are also under active investigation, e.g. Amano et al's handrail-gripping robot for firefighting [49], Ripin et al's pole climbing robot [118], Nechba, Xu,

Brown et al's "mobile space manipulator SM2" [77, 78], and Almonacid et al's parallel mechanism for climbing on pipe-like structures [73]. Truss climbing also has been acknowledged to have clear applications in inspection and construction of in-space structures [14]. Staritz et al's built Skyworker [86]. Kotay and Rus developed Inchworm" [54].

Chapter 3

Problem Formulation

The main contribution of this thesis is the theory for coordinated robotic construction, and this chapter shows setup and background for the theory as well as experiments implemented to prove the proposed algorithm.

3.1 Theory: Distributed Algorithms

We propose two distributed algorithms for construction by a team of networked robots: sub-assembly assignment and uniform delivery. Decentralized algorithms are essential so that they scale regardless of a number of robots and they are robust to a failure. The proposed algorithms are provably stable and convergent, and they turn out to be very adaptive to a failure of robots and dynamic constraints.

3.1.1 Domain

We are given a team of robots, n of which are specialized as assembling robots and the rest are specialized as part delivering robots in Euclidean space $Q \subset \mathbb{R}^N (N = 2, 3)$ or on a graph $G = (Q, E)$ where Q is a node set and E is an edge set. Let N_d be the number of delivery robots and N_a be the number of assembly robots. The robots can communicate locally with other robots within their communication range.

We differentiate the formulation according to the domain.

Euclidean space

The robots are given a target shape represented as a target density function $\phi_t : Q \to \mathbb{R}$. ϕ_t represents the goal shape geometry by specifying the intended density of construction material in space. For example, in Figure 3-1 the yellow region has high density (many materials) while the white region has low density. If the components can be built *independently* and an assembling robot is capable of assembling all of them, ϕ_t is linearly superposed as

$$\phi_t = \sum_{u=1}^{z} \beta_u \psi_u, \tag{3.1}$$

where z is the number of the components that can be assembled by an assembling robot, and β_u is a constant representing importance of the u^{th} component. Importance can measure time required to assemble the piece, time until the piece is needed in the assembly, etc.

Without loss of generality, we will focus the examples on truss structures built with two types of components: connectors and links in order to simplify exposition and figures. To represent truss structures, ϕ_t is defined point-wise on the grid that corresponds to the truss. The point density is proportional to the number of possible truss connection at the point. We wish to develop a decentralized algorithm that coordinates the robot team to deliver parts so that the goal assembly can be completed with maximum parallelism. We assume that the robots move *freely* in an Euclidean space (2D and 3D).

Graph

Suppose n robots cover an undirected graph $G = (Q, E)$ with the configuration $\{p_1, ..., p_n\}$, where $p_i \in Q$ is the position node of the i^{th} robot. $d(\cdot, \cdot) : E \to \mathbb{R}^+$ denotes the shortest distance measure between two nodes. $d(s, t) = \infty$ when s and t are not connected. The cost of an edge is strictly positive. Each node has a node-weight $\phi_t(q)$ denoting the importance of a task at q, which we call the target density function.

Next, we divide G into graph Voronoi partitions [30]. Given a node q in G, the nearest robot to q will execute the task at q. Each robot is allocated the task that includes its Voronoi partition V_i in G.

$$V_i = \{q \in Q | d(q, p_i) < d(q, p_j), \forall j \neq i\}. \tag{3.2}$$

Unlike Voronoi partitioning in a continuous space, we have to clarify the assignment of a node that has the same distance to multiple robots. We give priority to the robot with the minimum ID according to the following condition:

$$q \in V_i \Rightarrow i = \min \{j | d(q, p_i) = d(q, p_j)\}. \tag{3.3}$$

By adding weights to robots, we have a generalized Voronoi partition as given by:

$$V_i = \{q \in Q | d(q, p_i) - w_i < d(q, p_j) - w_j, \forall j \neq i\}, \tag{3.4}$$

where w_i is a weight. A larger weight yields a larger region.

To ensure distributed setting, we make the following assumptions.

1. The environment $(G, \phi_t(q))$ is given to each robot.
2. The node weight $\phi_t(q)$ is fixed.
3. The robots do *not* know the locations of the other robots.
4. The robots *do* precompute the distance matrix D of G as a $|Q| \times |Q|$ symmetric matrix where the matrix element d_{ij} is $d(q_i, q_j)$.

Because of the third assumption, the robots can not precompute the optimal configuration. The matrix D can be computed with $O(|Q|^3)$ runtime by the Floyd-Warshall algorithm [37].

3.1.2 Control algorithms

Algorithm 1 shows the main flow of construction in a *centralized* view. In the first phase, assembling robots spread in a convex and bounded target area Q which includes the target structure. They find placements using a distributed coverage controller which assigns to each robot areas of the target structure that have approximately the same assembly complexity. In the second phase the delivering robots move back and forth to carry source components to the assembling robots. They deliver their components to the assembling robot with maximum *demanding mass*. The demanding mass is defined as the amount of

a source component required for an assembling robot to complete its substructure. In this thesis, we restrict the source components include two types: unit-length truss elements and connectors. However, the algorithm is general and can support any number of different assembly components. After an assembling robot obtains a component from a delivering robot, it determines the optimal placement for this component in the overall assembly and moves there to assemble the component. The assembly phase continues until there is no source component left or the assembly structure is complete.

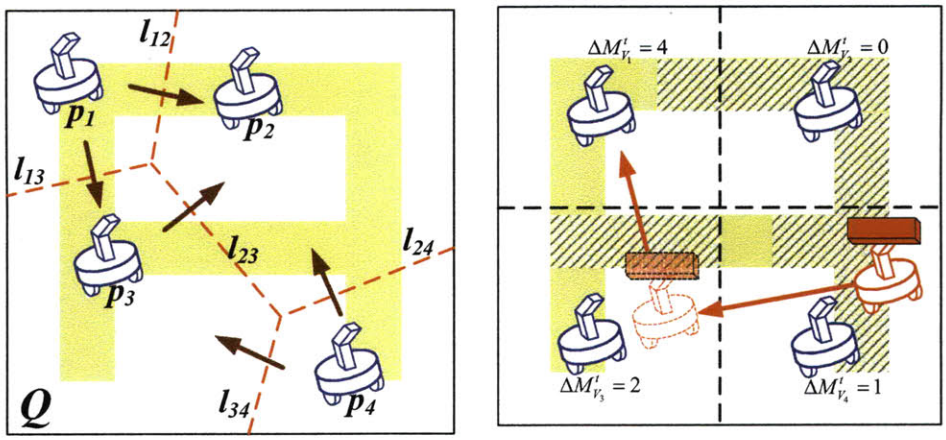
Algorithm 1 Construction Algorithm

1: Deploy the assembling robots in Q
2: Place the assembling robots at optimal task locations in Q (Chapter 4)
3: **repeat**
4: **delivering robots:** carry source components to the assembling robots
5: **assembling robots:** assemble the delivered components
6: **until** task completed *or* out of parts

3.1.3 Example

Figure 3-1(a) shows a construction system with 4 assembling robots. Intuitively, robot 1 and robot 4 move towards the other robots in order to expand their partition, whereas robot 2 moves away from the other robots because it has the largest area. The moving direction of the robots is determined by combining the normals to the Voronoi edges. Figure 3-1(b) shows the red delivering robot carrying a red truss element driven by the gradient of the demanding mass. The yellow region denotes the target density function ϕ_t (in Euclidean space). The hashed region denotes completed assembly. The demanding mass of a region can be thought of as the difference between the area of yellow regions and the area of hashed regions.

Suppose a delivering robot is in the region of robot 4. Among its neighbors (robot 2 and 3) the maximum demanding mass is with robot 3. Thus the delivering robot moves to robot 3. The delivering robot finds that robot 1 has the maximum demanding mass among robot 3's neighbors, therefore it advances to robot 1 and delivers the truss component. Following the maximum demanding mass gives a local balance for the target structure.



(a) (b)

Figure 3-1: Example of the equal-mass partitioning and delivery by the gradient of the demanding mass. 4 mobile manipulators (assembly robots) are displayed in a convex region Q that includes the A-shaped target structure. The yellow region has high density ϕ_t. The mass of a robot is the size of the total yellow region in its partition (Voronoi region). $p_i (i = 1, 2, 3)$ denotes the position of the assembling robots and the red-dotted lines l_{ij} are shared boundaries of the partitions between two robots. $\Delta M^t_{V_i}$ is the demanding mass.

3.1.4 Adaptiveness

Based on the assignment algorithm and the uniform delivery algorithm, we show their adaptation to failure of robots, dynamic constraints, multiple types of elements and reconfiguration. Continuous execution of the assignment algorithm for coverage during construction is a key to adaptiveness. While running the assignment algorithm, we can modify the density function so that it includes more information such as connectivity constrains and locations of trusses to be disassembled. Also, the algorithms can be used for general types of source elements.

3.2 System: Networked Robots

Our hardware system consists of a team of mobile manipulators, smart parts each with an embedded communication device, and a motion capture system. The robots operate on a square area, and a source cache is located at the end of the workspace (See Figure 8-16). Trusses and connectors are manually supplied to the cache during experiments. In

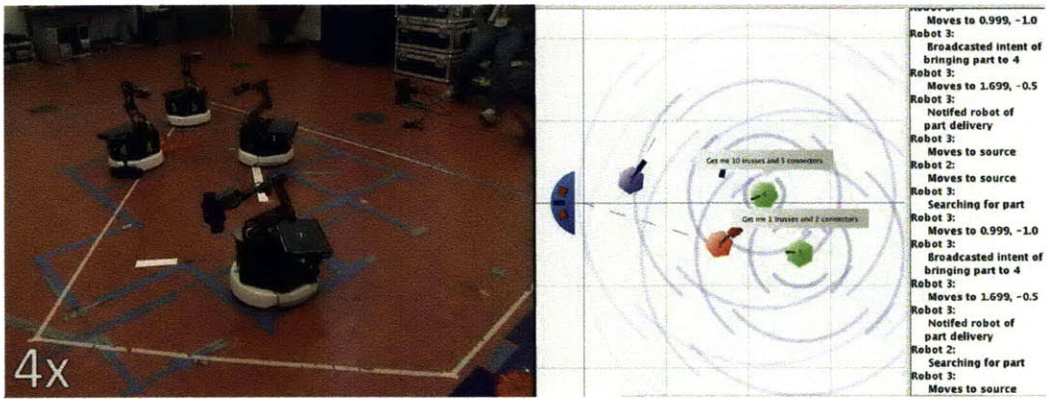


Figure 3-2: Snapshot of a delivery experiment. Robot 4 in the left has two orange connectors and one navyblue truss. The robot in the below is holding a truss and delivering. The GUI shows the status of the robots and communication.

order to help grasping, each 3D-printed smart part contains a custom IR chip and a battery designed to talk to the robots. The robots localize using data from the motion capture system broadcast over a mesh network.

Figure 3.2 shows a snapshot of a delivery experiment with two assembly robots and two delivery robots.

The details of the hardware are shown in Chapter 8.

Chapter 4

Subassembly Assignment: Equal-mass Partitioning

This section describes a decentralized equal-mass partitioning controller which is inspired by distributed coverage control [23, 100]. The algorithm allocates to each assembling robot the same amount of assembly work, which is encoded as the same number of truss elements. This condition ensures maximum parallelism. We continue with a review of the key notation in distributed coverage, then give the mass optimization criteria and end the section with the decentralized controller.

4.1 Distributed Coverage

Control of a robot group has become an important problem for robotics applications which cover obtaining a desired formation, optimizing sensing quality, maintaining a network connectivity, desirable sensory coverage, etc.

Based on applications, distributed coverage controllers optimize the cost and scope of sensing over the region. For example, to maximize the sensory coverage, it has been proven that the robots should minimize the following cost function $\mathcal{H}_0$ [23]:

$$\mathcal{H}_0 = \sum_{i=1}^{n} \int_{V_i} \frac{1}{2} \|\mathbf{q} - \mathbf{p}_i\|^2 \phi(\mathbf{q}) d\mathbf{q}, \tag{4.1}$$

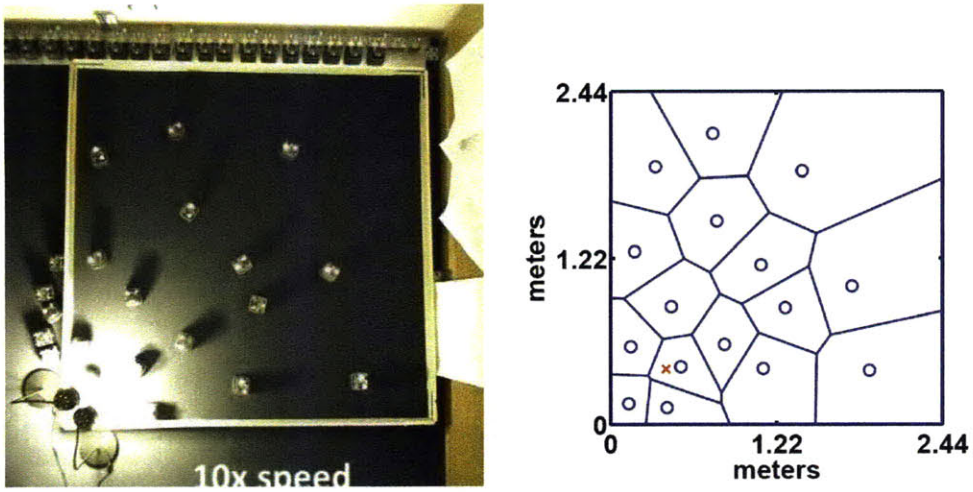


Figure 4-1: A group of SwarmBots are covering the square area to optimize sensing quality of the light intensity by the distributed coverage controller. Reprinted with permission from [99].

where $\phi(\mathbf{q})$ is a sensory function that corresponds to the target density function in this paper.

Figure 4-1 shows an example of distributed coverage in which a team of mobile robots react to the light intensity so that they can have a formation optimizing the sensory coverage.

Note that coverage implies dividing a target region into the same number of subregions as a number of robots.

4.2 Equal-mass partitioning

Suppose n assembling robots cover region Q with a configuration $\{\mathbf{p}_1, ..., \mathbf{p}_n\}$, where $\mathbf{p}_i$ is the position vector of the i^{th} robot. Given a point $\mathbf{q}$ in Q, the nearest robot to $\mathbf{q}$ will execute the assembly task at $\mathbf{q}$. Each robot is allocated the assembly task that included its Voronoi partition V_i in Q.

$$V_i = \{\mathbf{q} \in Q | \, \|\mathbf{q} - \mathbf{p}_i\| \leq \|\mathbf{q} - \mathbf{p}_j\| \, , \forall j \neq i\} \tag{4.2}$$

The target density function ϕ_t is the density of truss elements, and it is fixed during the construction phase. Given V_i, we define its mass property as the integral of the target

density function in the area.

$$M_{V_i} = \int_{V_i} \phi_t(\mathbf{q}) d\mathbf{q} \tag{4.3}$$

Distributed coverage controllers optimize the cost and scope of sensing over the region. It has been proven that the robots should minimize the following cost function $\mathcal{H}_0$ [23]:

$$\mathcal{H}_0 = \sum_{i=1}^{n} \int_{V_i} \frac{1}{2} \|\mathbf{q} - \mathbf{p}_i\|^2 \phi(\mathbf{q}) d\mathbf{q}, \tag{4.4}$$

where $\phi(\mathbf{q})$ is a sensory function that corresponds to the target density function in this thesis. We wish for each robot to have the same amount of assembly work. We call this *equal-mass partitioning*. The cost function can be modeled as the product of all the masses:

$$\mathcal{H} = \mathcal{H}_0 - \prod_{i=1}^{n} M_{V_i}, \tag{4.5}$$

where $\mathcal{H}_0$ is a constant and the bound of the product term as:

$$\mathcal{H}_0 = \left(\frac{1}{n} \sum_{i=1}^{n} M_{V_i} \right)^n = \left(\frac{1}{n} \int_Q \phi_t(\mathbf{q}) d\mathbf{q} \right)^n. \tag{4.6}$$

The cost function is continuously differentiable since each M_{V_i} is continuously differentiable [87]. Minimizing this cost function leads to equal-mass partitioning, because of the relationship between the arithmetic mean and the geometric mean.

$$\frac{1}{n} \sum_{i=1}^{n} M_{V_i} \geq \sqrt[n]{\prod_{i=1}^{n} M_{V_i}}, \tag{4.7}$$

where the equality holds only if all the terms are the same. Therefore the prefect equal-mass partitioning makes the cost function zero. Using the cost function in (4.5), we have developed a decentralized controller that guarantees $\mathcal{H}$ converges to a local minimum.

4.2.1 Controller with Guaranteed Convergence

We wish for the controller to continuously decrease the cost function: $\dot{\mathcal{H}} \leq 0,\ t > 0$. Differentiating $\mathcal{H}$ yields

$$\dot{\mathcal{H}} = \sum_{i=1}^{n} \frac{\partial \mathcal{H}}{\partial \mathbf{p}_i} \dot{\mathbf{p}}_i. \tag{4.8}$$

When $\mathcal{N}_i$ is a set of neighbor robots of the i^{th} robot, each term of the partial derivatives is

$$\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i} = -\sum_{j=i,\mathcal{N}_i} \frac{\partial M_{V_j}}{\partial \mathbf{p}_i} \prod_{k=\{1,\ldots,n\},k\neq j} M_{V_k} \tag{4.9}$$

$$= -\prod_{l\notin\{i,\mathcal{N}_i\}} M_{V_l} \sum_{j=i,\mathcal{N}_i} \frac{\partial M_{V_j}}{\partial \mathbf{p}_i} \prod_{k\in\{i,\mathcal{N}_i\},k\neq j}^{n} M_{V_k} \tag{4.10}$$

where

$$\frac{\partial M_{V_i}}{\partial \mathbf{p}_i} = \sum_{j\in\mathcal{N}_i} \mathcal{M}_{ij}, \quad \frac{\partial M_{V_j}}{\partial \mathbf{p}_i} = -\mathcal{M}_{ij} \tag{4.11}$$

$\mathcal{M}_{ij}$ is computed along the sharing edges (sharing faces in 3D) l_{ij} between V_i and V_j as in [87]:

$$\mathcal{M}_{ij} = \int_{l_{ij}} \phi_t(\mathbf{q}) \frac{\partial \mathbf{q}_{l_{ij}}}{\partial \mathbf{p}_i} \mathbf{n}_{l_{ij}} d\mathbf{q} = \int_{l_{ij}} \phi_t(\mathbf{q}) \frac{\mathbf{q}-\mathbf{p}_i}{\|\mathbf{p}_i-\mathbf{p}_j\|} d\mathbf{q} \tag{4.12}$$

where $\mathbf{n}_{l_{ij}}$ is a normal vector to l_{ij} as

$$l_{ij} = V_i \cap V_j, \quad \mathbf{n}_{l_{ij}} = \frac{\mathbf{p}_j-\mathbf{p}_i}{\|\mathbf{p}_i-\mathbf{p}_j\|}. \tag{4.13}$$

Figure 4-2 shows the notion used in this thesis. We can rewrite equation 4.8 as

$$\dot{\mathcal{H}} = -\sum_{i=1}^{n} \prod_{l\notin\{i,\mathcal{N}_i\}} M_{V_l} \sum_{j=i,\mathcal{N}_i} \frac{\partial M_{V_j}}{\partial \mathbf{p}_i} \prod_{k\in\{i,\mathcal{N}_i\},k\neq j}^{n} M_{V_k} \dot{\mathbf{p}}_i \tag{4.14}$$

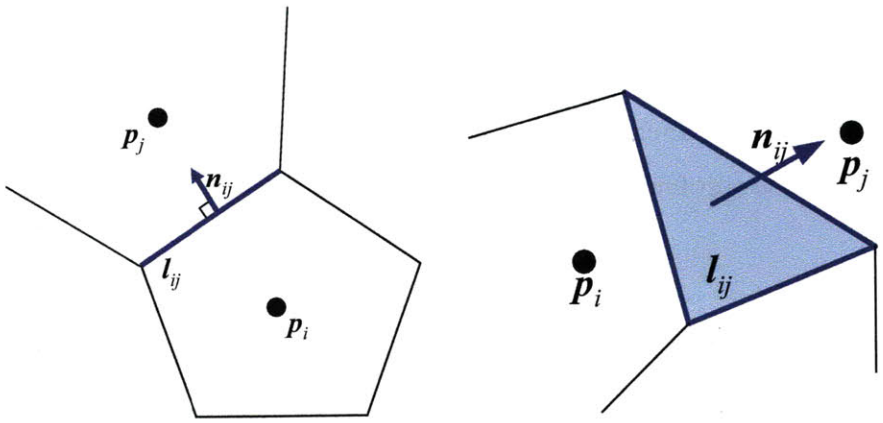


Figure 4-2: Between two Voronoi regions(2D and 3D), l_{ij} and n_{ij} are defined as in these figures.

Let $\boldsymbol{J}_i$ denote the part of the partial derivative term $\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i}$ which is related with the set $\{i, \mathcal{N}_i\}$.

$$\boldsymbol{J}_i = \sum_{j=i,\mathcal{N}_i} \frac{\partial M_{V_j}}{\partial \mathbf{p}_i} \prod_{k \in \{i,\mathcal{N}_i\}, k \neq j}^{n} M_{V_k} \tag{4.15}$$

Note that $\boldsymbol{J}_i$ is a vector. Given a velocity control for each robot, the decentralized controller that achieves task allocation is given by the control law:

$$\dot{\mathbf{p}}_i = k \frac{\boldsymbol{J}_i}{\|\boldsymbol{J}_i\|^2 + \lambda^2}, \tag{4.16}$$

where k is a positive control gain and λ is a constant to stabilize the controller even around singularities where $\|\boldsymbol{J}_i\|^2 = 0$.

Note that all the equations can be computed in a distributed way, since they only depend on the variables of the neighboring robots.

Theorem 1 *The proposed controller guarantees that $\mathcal{H}$ converges to either a local maximum or a global maximum.*

Proof: The proposed control input $\dot{\mathbf{p}}_i$ yields

$$\dot{\mathcal{H}} = -k \sum_{i=1}^{n} \frac{\|\boldsymbol{J}_i\|^2}{\|\boldsymbol{J}_i\|^2 + \lambda^2} \prod_{l \notin \{i,\mathcal{N}_i\}} M_{V_l}. \tag{4.17}$$

Since k and M_{V_i} are positive, each term of $\dot{\mathcal{H}}$ is always negative. In addition, the cost function is differentiable, and trajectories of robots are bounded in Q. Therefore, the controller keeps the cost function decreasing unless all the $\boldsymbol{J}_i$ are empty vectors (relocating the robots does not change the cost function), which implies a local minimum.[1] □

4.2.2 Equal-mass Partitioning with Locational Optimization

Although the proposed controller leads the robots to equal-massed regions, the shapes of the regions may not always look intuitively right. For example, if the controller divides a square Q with a uniform density function, it may give a set of vertically long strips rather than a set of squares that is more desirable in a sense of traveling time and communication range. Fortunately, the balanced region with respect to a density function can be thought as the locational optimization with the cost function $\mathcal{H}_0$, since minimizing uncertainty leads to compact regions.

We add the locational optimization property to the equal-mass partitioning controller. We re-define $\mathcal{H}_0$ as:

$$\mathcal{H}_0 = \sum_{i=1}^{n} M_{V_i} \left\| \mathbf{C}_{V_i} - \mathbf{p}_i \right\|^2. \tag{4.18}$$

since a solution of the locational optimization is to locate robots at their centroid $\mathbf{C}_{V_i}$ [23]. Differentiating the cost function with respect to $\mathbf{p}_i$ yields [87]:

$$\frac{\partial \mathcal{H}_0}{\partial \mathbf{p}_i} = -2(\mathbf{L}_{V_i} + \mathbf{R}_i - M_{V_i}\mathbf{p}_i) \tag{4.19}$$

where

$$\mathbf{R}_i = \sum_{j \in \mathcal{N}_i} \left[\frac{1}{2}\mathcal{M}_{ij}(\mathbf{C}_{V_i}^T\mathbf{C}_{V_i} - \mathbf{C}_{V_j}^T\mathbf{C}_{V_j}) - \mathcal{L}_{ij}(\mathbf{C}_{V_i} - \mathbf{C}_{V_j}) \right] \tag{4.20}$$

and

$$\mathcal{L}_{ij} = \int_{l_{ij}} \phi_t(\mathbf{q})\mathbf{q} \left(\frac{\mathbf{q} - \mathbf{p}_i}{\|\mathbf{p}_i - \mathbf{p}_j\|} \right)^T d\mathbf{q}. \tag{4.21}$$

[1] Pavone et. al [85] also developed equitable partitioning using power diagrams that are weighted generalized Voronoi diagrams. They used a different cost function as the average of inverse of the masses. They targeted a different application in the space of the multi-vehicle routing.

We can re-set the final cost function as a linear combination of $\mathcal{H}$ and $\mathcal{H}_0$ as:

$$\hat{\mathcal{H}} = \mathcal{H} + \gamma\mathcal{H}_0 \tag{4.22}$$

where γ is a positive constant that can be tuned. Differentiating this with respect to time gives:

$$\dot{\hat{\mathcal{H}}} = \sum_{i=1}^{n}(\frac{\partial\mathcal{H}}{\partial\mathbf{p}_i} + \gamma\frac{\partial\mathcal{H}_0}{\partial\mathbf{p}_i})\dot{\mathbf{p}}_i. \tag{4.23}$$

Now we have a new $\hat{\boldsymbol{J}}_i$:

$$\hat{\boldsymbol{J}}_i = \sum_{j\in\mathcal{N}_i}(M_{V_i} - M_{V_j})(\frac{\partial M_{V_i}}{\partial\mathbf{p}_i} - \frac{\partial M_{V_j}}{\partial\mathbf{p}_i}) - 2\gamma(\mathbf{L}_{V_i} + \mathbf{R}_i - M_{V_i}\mathbf{p}_i). \tag{4.24}$$

If the control input is set:

$$\dot{\mathbf{p}}_i = -\frac{k}{4}\frac{\hat{\boldsymbol{J}}_i}{\left\|\hat{\boldsymbol{J}}_i\right\|^2 + \lambda 2}\left[\sum_{j\in\mathcal{N}_i}(M_{V_i} - M_{V_j})2 + \gamma M_{V_i}\left\|\mathbf{C}_{V_i} - \mathbf{p}_i\right\|^2\right], \tag{4.25}$$

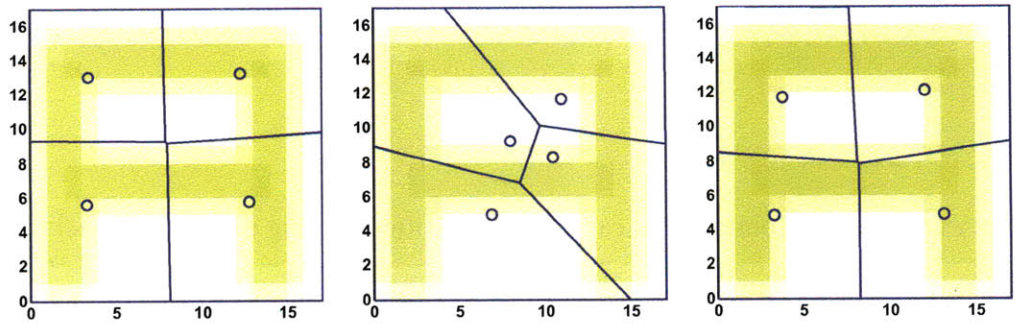
we have the same convergence property for $\hat{\mathcal{H}}$.

Figure 4-3 shows Vornoi regions from the three controllers: locational optimization, equal-mass partitioning, combined controller. The equal-mass partitioning only gives skewed regions although each region has the same mass. The combined controller shows the best result in terms of locationally balanced regions and equal-mass. From now on, we denote equal-mass partitioning as the combined controller.

4.2.3 Implementation

The *equal-mass partitioning* was implemented for building 2D and 3D structures. We use side truss elements and connectors that lie at a single source location. We have built several structures using these algorithms.

The first simulation demonstrates the construction of a bridge from a single source location of trusses and connectors. The density function ϕ_t and the final Voronoi regions resulting from using the equal-mass partitioning controller for 4,6, and 10 assembling robots are



(a) (b) (c)

Figure 4-3: Density function for an A-shaped bridge and resultant Voronoi regions. The blue circles are assembling robots. Yellow regions have dense ϕ_t. (a) locational optimization only (b) the equal-mass partitioning only (c) the combined controller

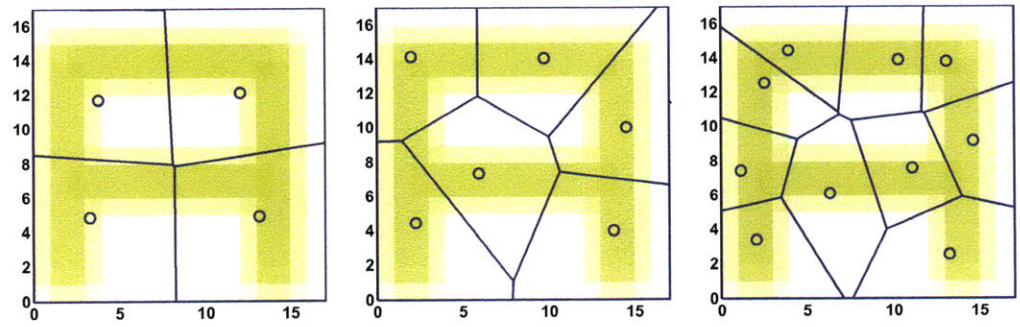
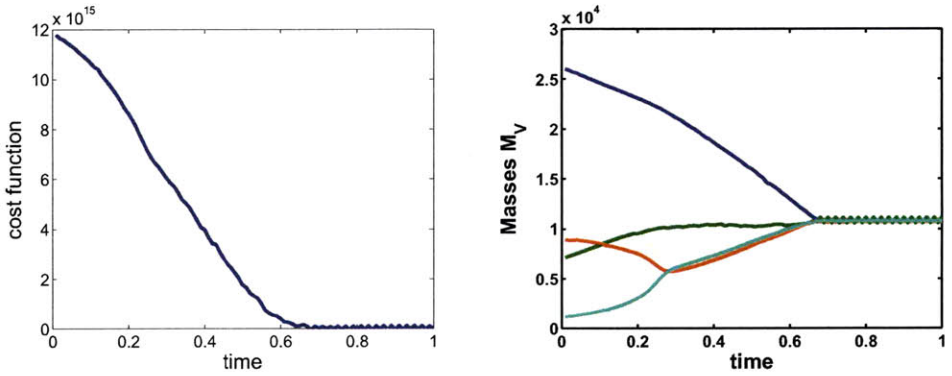


Figure 4-4: Density function for an A-shaped bridge and coverage by the equal-mass partitioning. The blue circles are assembling robots. Yellow regions have dense ϕ_t.

shown in Figure 4-4. We use a discrete system so that ϕ_t is defined at every node (integer points). The unit length is the length of a truss element. At an arbitrary point $\mathbf{q}$, $\phi_t(\mathbf{q})$ is interpolated from 4 surrounding nodes by barycentric interpolation. The interpolation ensures continuity of ϕ that is required for the cost function $\mathcal{H}$. The robots are deployed from randomly selected starting positions. Figure 4-4 shows that each robot has approximately the same area of the yellow region. As expected, the masses converge to the same value as shown in Figure 4-5(b), and the cost function $\mathcal{H}$ approaches zero as in Figure 4-5(a). A little jitter in the masses and the cost function graphs comes from discrete numerical integrals.



(a) (b)

Figure 4-5: Result from the equal-mass partitioning controller for 4 assembling robots. (a) Cost function $\mathcal{H}$ (b) Masses of four assembling robots

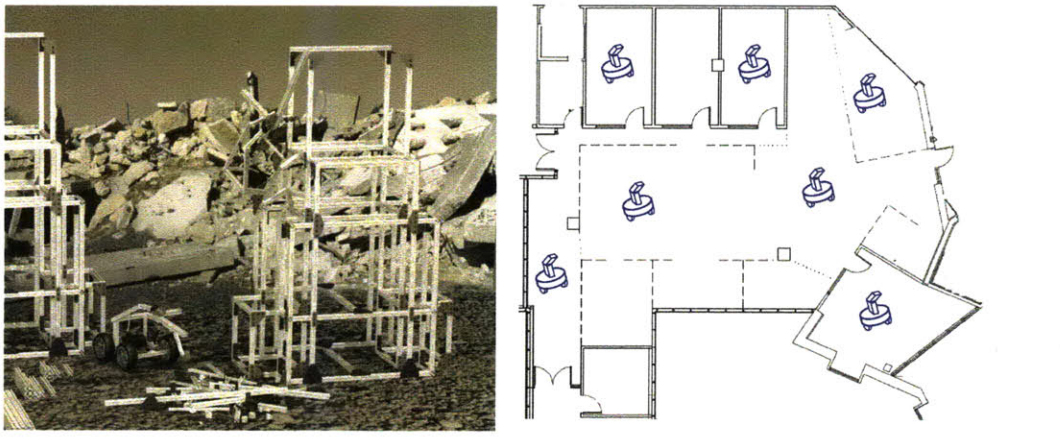
Chapter 5

Subassembly Assignment: Distributed Partitioning on a Graph

5.1 Coverage on a Graph

In distributed coverage on a graph, a set of mobile robots uses local information to place themselves in such a way as to optimally cover the nodes of the graph according to some problem-specific metric. The robots, the environment, and the actions in such a system are all discrete. For example, we can apply this method to decentralized construction (see Figure 5-1(a)) where robots cooperate to assemble a complex structure out of discrete components, possibly by dividing it into subassemblies. Most decentralized coverage solutions for continuous spaces work with convex environments ([23, 100, 85]), but many indoors and outdoors environments are not convex (for example see Figure 5-1(b)). Decentralized coverage on a graph can be used to provide an approximated solution to continuous non-convex domains by modeling the non-convex region by a mesh network as in finite element methods (FEM).

Decentralized coverage on a graph for problem domains characterized by discrete structure such as construction, transportation and facilities planning has significant advantages over using its continuous counterpart in which the discrete structure is modeled as a continuous density function. The advantages are:



(a) (b)

Figure 5-1: Applications of distributed coverage on graph. (a) Concept art for construction of a truss structure by mobile delivering robots and truss-climbing assembling robots. Reprinted with permission from Jonathan Hiller, Cornell University, USA. (b) Coverage of a complicated non-convex region. The blue robots are covering the 3rd floor of Stata Center at MIT.

1. each coverage region ensures connectivity, while coverage computed with continuous methods adapted to the discrete domain may lead to disconnected subsets of the graph within a single region.

2. robot neighbors are explicitly connected by edges, while neighbors computed by continuous coverage methods adapted to the graph domain may not be physically reachable.

3. The graph can be a representation of a non-convex region.

Also, a robot failure can be handled transparently as in continuous domains [23].

The trade-off is the discrete method requires more computation than the continuous approach especially when the number of nodes is large but a corresponding continuous density function has good properties for the computation such as numerical integration. As we will see in Section 5.2, the runtime of our algorithm has order of square of the node number.

Next, we describe the partitioning problems studied in this thesis: locational optimization and equal-mass partitioning. We focus on solving the locational optimization, and the solution is extended for equal-mass partitioning.

5.1.1 Locational optimization

Locational optimization has been extensively researched in operations research for a variety of optimization problems such as placing facilities to minimize costs (distances). For example, how should we locate post offices to minimize the total distance from inhabitants in the area? Recently the locational optimization was revisited in robotics and control, for distributed coverage of multi-robot systems ([23, 100]). In distributed coverage, a team of robots cover an area of interest to optimize a cost function.

In graph theory, this problem is called p-median (not to be confused with the standard way of denoting the position of a robot by variable $\mathbf{p}$). The goal is to find the best set of medians (centroids) of the given graph. The cost function is given as:

$$\mathcal{H}_L = \sum_{i=1}^{n} \sum_{V_i} \phi_t(q) d(q, p_i), \tag{5.1}$$

which is the discretized cost function used in locational optimization [23]:

$$\mathcal{H}_0 = \sum_{i=1}^{n} \int_{V_i} \phi_t(\mathbf{q}) \left\| \mathbf{p}_i - \mathbf{q} \right\|^2 d\mathbf{q}, \tag{5.2}$$

where $\mathbf{q}$ and $\mathbf{p}_i$ are now position vectors.

The p-median problem is NP-hard for a non-tree graph [40]. A great number of heuristic centralized solutions have been proposed [91]. Our approach implements distributed coverage of multi-robot system on a graph and is new in that it provides:

1. a distributed controller for a mobile robot system,

2. a geometry-based solution using graph Voronoi tessellation.

5.2 Decentralized Control Algorithms for Locational Optimization

In this section, we propose decentralized controllers to achieve locational optimization. We will extend the controllers for equal-mass partitioning in Section 5.3.

Algorithm 2 shows the main control loop. Each robot has two states:

- *COMPUTE*: compute the optimal node to relocate
- *MOVING*: move to the optimal node.

Algorithm 2 Distributed Controller

STATE: *COMPUTE*

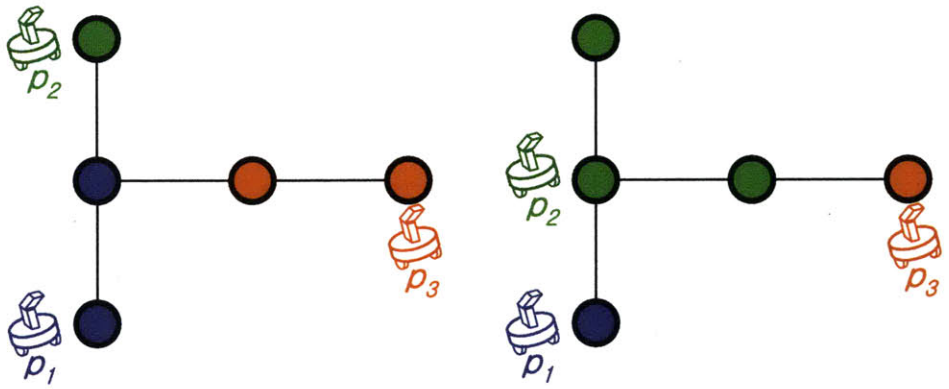
1: Communicate with $\mathcal{N}_i$
2: Construct a new Voronoi partition by $\left\{p_i^*, p_{\mathcal{N}_i}^*, p_{\mathcal{N}_{\mathcal{N}_i}}^*\right\}$
3: Find the new optimal p_i^* (Algorithm 3, 4)
4: **if** $p_i \neq p_i^*$ **then**
5: state = MOVING
6: **end if**

STATE: *MOVING*

7: Move to p_i^*
8: **if** $p_i = p_i^*$ **then**
9: state = FIND
10: **end if**

In contrast to the distributed coverage controller in a continuous domain [23] where each robot requires only information about its neighbors $\mathcal{N}_i$, in the graph case each robot needs to know information about all the neighbors of its neighbors $\mathcal{N}_{\mathcal{N}_i}$ as well. This is because relocation of the robot on a graph can not be infinitely small as in the controllers for the continuous domain. Therefore the relocation may change the Voronoi region of $\mathcal{N}_{\mathcal{N}_i}$. Figure 5-2 shows an example. The initial graph's Voronoi tessellation is shown in Figure 5-2(a). Each color represents its Voronoi region V_i. All the edges are unit distance long. If robot 2 moves downward by an edge as in Figure 5-2(b), V_3 changes although robot 3 was not a neighbor of robot 1 in the initial configuration.

In the *COMPUTE* state, the robot communicates and receives information about the neighbors $\mathcal{N}_i$ and the neighbors of its neighbors $\mathcal{N}_{\mathcal{N}_i}$ to construct the graph Voronoi tessellation. p_i^* is the centroid of V_i denoting the desired location for robot i. Note that p_i^* can be different from the actual position p_i while the robot is moving to p_i^* in the *MOVING* state. Therefore, the graph Voronoi tessellation should be built from the set of p^*, not from p. After building the current Voronoi tessellation, each robot determines the optimal node for its relocation. Algorithm 3 finds the optimal node for locational optimization. The algorithms



(a) (b)

Figure 5-2: An example shows why a robot needs to know information of neighbors of the neighbors $\mathcal{N}_{\mathcal{N}_i}$. Each node and edge have unit weight and cost. Colors of nodes denote which robot they belong to.

guarantee decay of the cost functions. If the found p_i^* is not p_i, the robot switches to the state *MOVING* and moves to p_i^*.

5.2.1 Why 2-hop information?

Intuitively, we need 2-hop communication because we need to access the location of the neighbors' neighbors. This can be done by 2-hop communication (contact the neighbors to get their neighbors' locations). Alternatively, we can store the location of the neighbors with each node and use 1-hop communication. The trade-off is that as the neighbors move many updates (hence communications) may be necessary.

More specifically, we want relocation of robot i to change only the Voronoi regions of itself and its neighbors, so that we can decouple the cost function as follows:

$$\mathcal{H}_L = \mathcal{H}_i + \mathcal{H}_{\setminus i},$$

where

$$\mathcal{H}_i = \sum_{l=i, l\in\mathcal{N}_i} \sum_{V_i} \phi_t(q) d(q, p_i),$$

and

$$\mathcal{H}_{\setminus i} = \mathcal{H}_L - \mathcal{H}_i.$$

$\mathcal{H}_i$ is a part of the cost function that can be changed by the relocation of robot i. It includes only V_i and $V_{\mathcal{N}_i}$, where V_i is Voronoi partition of robot i. We want the remaining part $\mathcal{H}_{\setminus i}$ untouched while robot i is moving. To ensure this decoupling, robot i should know the locations of $\mathcal{N}_{\mathcal{N}_i}$ (neighbors of the neighbors). Note that we have shown that the relocation of a robot may change the Voronoi region of $\mathcal{N}_{\mathcal{N}_i}$. Given the locations $\mathcal{N}_{\mathcal{N}_i}$, the proposed *distributed vertex substitution* algorithm ensures no change in $\mathcal{N}_{\mathcal{N}_i}$.

Without 2-hop information, we can not decouple the cost function.

Next we explain the details of the algorithms for locational optimization. The algorithm is based on *vertex substitution*. Vertex substitution [107] is known as a typical solution for the p-median problem [91]. We modify it to fit our problems and call the modified version *distributed vertex substitution*.

5.2.2 Distributed vertex substitution algorithm

Algorithm 3 shows the distributed vertex substitution algorithm for locational optimization. Given the position set $P_i = \left\{p_i^*, p_{\mathcal{N}_i}^*, p_{\mathcal{N}_{\mathcal{N}_i}}^*\right\}$, let D_i^Q be a $|Q| \times |P_i|$ sub-matrix of D with all rows and columns matching P_i.

Among the nodes q_b in V_i, the algorithm finds the optimal node to substitute the current position. The algorithm checks how the substitution will affect the Voronoi tessellation and the cost function, by examining how $q_j \in \{V_i \cup V_{\mathcal{N}_i} \cup B_{\mathcal{N}_i}\}$ will change. $B_{\mathcal{N}_i}$ is a node set that does not belong to but is connected to $V_{\mathcal{N}_i}$. Therefore, it represents the nodes of $\mathcal{N}_{\mathcal{N}_i}$ that can be affected. We consider two cases for q_j: whether q_j will belong to the robot or not.

If so, substitution of p_i^* by q_b may lead to three different situations:

1. the new distance $d(q_j, q_b)$ is closer to the current distance $d(q_j, p_i)$
2. $d(q_j, q_s) > d(q_j, q_b) > d(q_j, p_i)$
3. $d(q_j, q_b) > d(q_j, q_s) > d(q_j, p_i)$.

For the first two cases, the cost function decreases by $\phi_t(q_j)(d_{jb} - d_{ji})$, whereas, for the last case, the cost function increase it by $\phi_t(q_j)(d_{js} - d_{ji})$. The amount of change to the cost function is denoted as ${}_j\Delta_{bi}$.

If q_j will not belong to the robot, the cost function increases by $\phi_t(q_j)(d_{js} - d_{ji})$ only when the closest robot r_k to q_j is in $\mathcal{N}_i$. If $r_k \notin \mathcal{N}_i$, we may change $V_{\mathcal{N}_{\mathcal{N}_i}}$. Therefore we do not consider q_b as a substitute for p_i^*. This guarantees the algorithm will only change the neighboring Voronoi regions.

The final node for substitution is chosen to reduce the cost function most among all Δ_{bi}. Δ_b, the minimum of Δ_{bi}, must be negative. Otherwise, the algorithm returns null, that is, the robot i does not move.

5.2.3 Analysis

The runtime of Algorithm 3 is $O(n\,|Q|^2)$ due to the two loops. We expect this algorithm to be used when the number of nodes in the graph is much larger than the number of robots. In such a case the running time will be dominated by the term due to the size of the node set and can be considered to be $O(|Q|^2)$.

Given the locations of the robots connected by a 2-hop communication, we prove Algorithms 3 and 4 convergence to local minima.

Let Ω be a set of all the possible configuration with n robots.

Theorem 2 *Ω is a bounded and invariant set.*

Proof: The number of the possible configuration is $\binom{|Q|}{n}$, where $|Q|$ is the number of nodes in G. Therefore Ω has a finite number of configurations, and it is bounded. Also, it is invariant since it contains every possible set. □

Let $M \subset \Omega$ be the set of critical configurations in which the robots do not reconfigure any more ($\Delta\mathcal{H}_L = 0$), given the distributed vertex substitution control algorithm.

Theorem 3 *M is an invariant set.*

Proof: Given the controller, robots do not move when $\Delta\mathcal{H}_L = 0$. Therefore, once a configuration yields $\Delta\mathcal{H}_L = 0$, this configuration remains constant. □

Algorithm 3 Distributed Vertex Substitution Algorithm for Locational Optimization

$D_i^Q = D(:, p_i^* \cup p_{\mathcal{N}_i}^* \cup p_{\mathcal{N}_{\mathcal{N}_i}}^*)$
for $q_b = \{q \in V_i | q \neq p_i\}$ **do**
  for $q_j = \{q \in V_i \cup V_{\mathcal{N}_i} \cup B_{\mathcal{N}_i} | q \neq p_{\mathcal{N}_i}\}$ **do**
    $d_{jk} \leftarrow$ min(row(D_i^Q, j))
    $r_k \leftarrow$ ID of the robot at k
    $d_{js} \leftarrow$ 2nd smallest row(D_i^Q, j)
    if $r_i = r_k$ **then**
      if $d_{jb} \leq d_{ji}$ or $(d_{jb} > d_{ji}$ and $d_{js} > d_{jb})$ **then**
        ${}_j\Delta_{bi} = \phi_t(q_j)(d_{jb} - d_{ji})$
      else
        ${}_j\Delta_{bi} = \phi_t(q_j)(d_{js} - d_{ji})$
      end if
    else
      if $d_{jb} < d_{jk}$ **then**
        if $r_k \notin \mathcal{N}_i$ **then**
          discard q_b
        end if
        ${}_j\Delta_{bi} = \phi_t(q_j)(d_{jb} - d_{jk})$
      end if
    end if
  end for
  $\Delta_{bi} = \sum_{j=1}^{|Q_j|} {}_j\Delta_{bi}$
end for
$\Delta_b = \min \Delta_{bi}$
if $\Delta_b \geq 0$ **then**
  return $\varnothing$
else
  return q_{b^*}
end if

Theorem 4 *Every configuration in Ω converges to M.*

Proof: Let $\mathcal{H}_0$ be the minimum of $\mathcal{H}_L$ in Ω. A configuration with $\mathcal{H}_0$ is the global optimum. $\mathcal{H}_0$ is the lower bound of $\mathcal{H}_L$, and the configuration with $\mathcal{H}_0$ should be in M.

Let ϵ be the *smallest negative change* in $\mathcal{H}_L$ between every possible pair of configurations in Ω. Since Ω has a finite number of configurations, ϵ is also finite. Therefore, given any configuration with the proposed controller, $\Delta\mathcal{H}_L$ is either 0 or less than ϵ. If $\Delta\mathcal{H}_L = 0$, the configuration is in M. If not, the configuration converges to M within a finite number of runs $T < \frac{\mathcal{H}_L - \mathcal{H}_0}{\epsilon}$, because the cost function decreases at least by ϵ and it is lower bounded by $\mathcal{H}_0$.

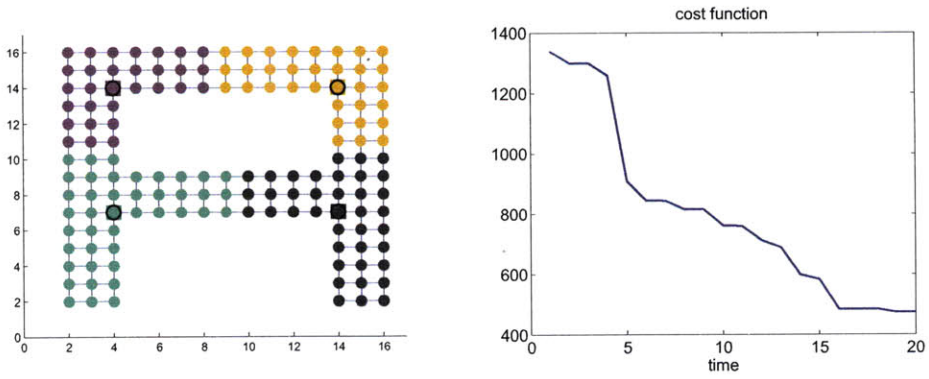
Since M is invariant, any configuration in Ω converges to the critical configuration. □

5.2.4 Implementation

We implemented Algorithms 2 and 3 and tested them on a suit of graph topologies. We focus on reporting on two graphs with similar topology but different sizes representing blueprints of bridge structures. The first structure shown in Figure 5-3(a) has 144 nodes and 240 edges. The second structure is shown in Figure 5-5(a). It has 384 nodes and 649 edges. 2$\sim$10 robots are tested for the small bridge, while 2$\sim$15 robots are simulated for the big bridge. Each set of robots is simulated 20 times with randomly initialized configurations. The simulations terminate when the Voronoi tessellation does not change after an iteration of the robot control loop.

Note that the first structure represents a mesh-network of the non-convex shape with the uniform density function shown as the yellow region in Figure 5-4. Therefore the resultant partitions are approximated partitions from continuous distributed coverage which does not exist yet. The finer mesh network will yield more precise approximation.

Figure 5-3(a) shows the resultant graph Voronoi regions of the small bridge obtained by the proposed controller for locational optimization with 4 robots. The graph of the cost function is shown in Figure 5-3(b). We see that the cost function decreases over time. By comparison, Figure 5-4 shows the solution computed by a e distributed coverage controller for a continuous domain, where robots move not only on the target structure but also in



(a) (b)

Figure 5-3: Simulation result from coverage on the small bridge. Nodes are filled circles and edges are black solid lines connecting the nodes. The circles enclosed by the black outline are robot locations. Each color represents a Voronoi region that belongs to the same colored robot. (a) The final configuration of locational optimization on the small bridge by 4 robots. (b) The cost function $\mathcal{H}_L$.

free space (white region). The weighting function is continuously defined by interpolating the node weights. The distributed controller in [23] is used for distributed coverage in Figure 5-4. The cost function for the continuous domain is shown in Equation 5.2. The final locations of the robots look almost identical. However, we can clearly see our algorithm for graph coverage ensures fully connected V_i and neighbors whose regions are physically connected. The distributed controller in the continuous domain may result in V_i with separated parts and physically non-connected neighbors. In Figure 5-4, you can see the upper-right robot and the lower-left robot are neighbors, although they are not connected by the target structure.

Figure 5-5(a) shows the final Voronoi regions from Algorithm 3 controllers with 15 robots on the big bridge. The result matches our intuition to locate the robots at the joints of the bridge.

We used two centralized methods to compare our solution with centralized solutions capable of computing global optima: integer programming [92] and Lagrangian relaxation heuristics [94]. The first approach could not handle the problem complexity. We used MATLAB and SCIP 1.2.0 [2] on 64bit Quad CPU Q9550. The software failed to compute the global optimum even for the smaller graph in Figure 5-3(a), because the computation load was too high.

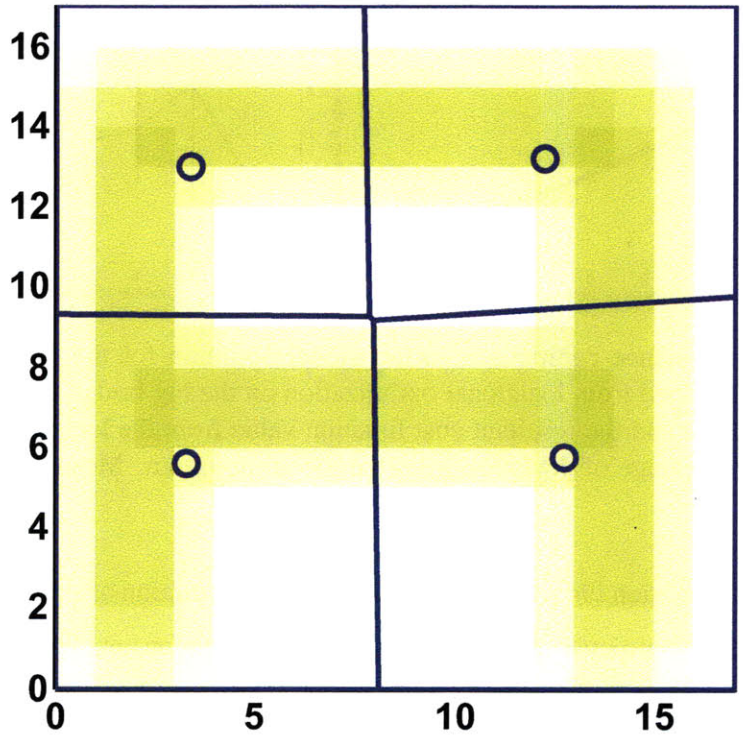
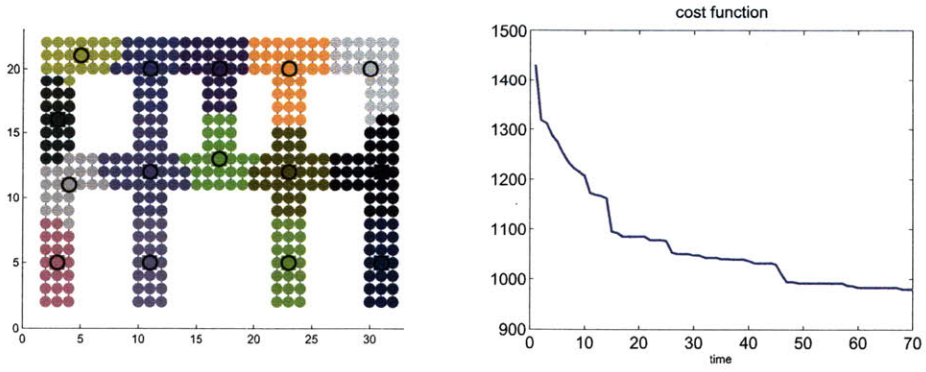


Figure 5-4: The resultant Voronoi regions by locational optimization with the continuous density function. The yellow region denotes high-density area while the white region has low density. The blue circles are robots.



(a) (b)

Figure 5-5: Simulation result from 15 robot coverage on the big bridge. (a) The final configuration of locational optimization on the big bridge. (b) The cost function $\mathcal{H}_L$.

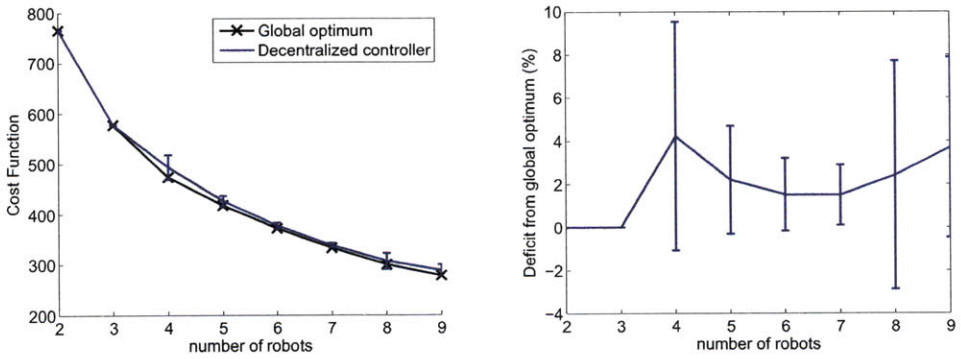


Figure 5-6: Performance comparison to the global optimum for the graph in Figure 5-3(a). Data are obtained from locational optimization on the big bridge by 2-15 robots. (a) the global optimum and the resultant cost function value from the locational optimization controller (b) Percentage of deviation from the global optimum. Mean and error-bars are shown.

Lagrangian relaxation [94] does not guarantee the computation of a global optimum but outputs whether its computed solution is the global optimum or not. We used this method to evaluate all the solutions computed in a distributed way by our method. Lagrangian relaxation produces the global optimums 8 times out of 9 cases for the graph in Figure 5-3(a), and 7 of 14 cases for the graph in Figure 5-5(a).

Figures 5-6 and 5-7 show the results of using Lagrangian relaxation (centralized method) to compute the global optima. Deviation from the guessed global minimum slowly increases as number of the robots increases, however it remains within 10%.

We are continuing research on centralized computationally tractable methods for identifying the global optimum and evaluating our algorithm on more general test sets such as OR library [12].

5.3 Extension to equal-mass partitioning

Using a different cost function, we can extend Algorithms 2 and 3 for a related problem: equal-mass partitioning. This problem is important in decentralized construction where we seek to identify subassemblies that can be aggregated in approximately the same time period [65] and in vehicle routing where we want each vehicle to cover the same workload

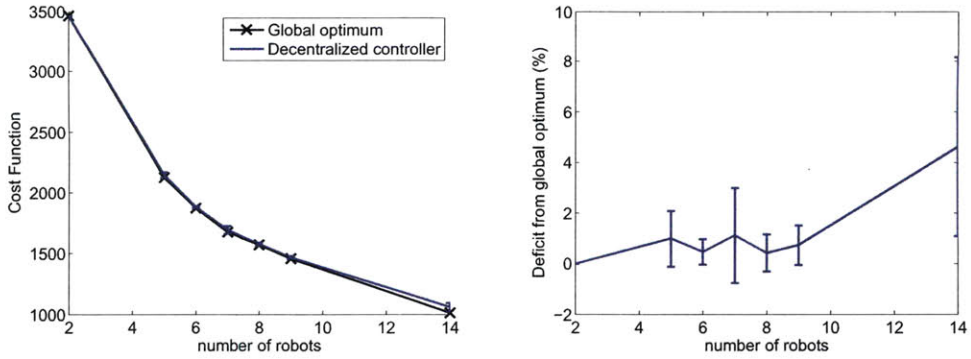


Figure 5-7: Performance comparison to the global optimum for the graph in Figure 5-3(b). Data are obtained from locational optimization on the big bridge by 2-15 robots. (a) the global optimum and the resultant cost function value from the locational optimization controller (b) Percentage of deviation from the global optimum. Mean and error-bars are shown.

in its route [85]. We introduce the problem, describe the distributed algorithm, and evaluate the algorithms for the same two graph topologies used to evaluate Algorithm 2 and 3

5.3.1 Equal-mass partitioning

The equal-mass partitioning problem is to divide an area of interest into pieces with equal amount of workload. It is useful for balancing the workload in multi-agent systems. Mass can be viewed as the physical measure of the weight associated with each region, or as an abstract measure. Solutions have been proposed based on a centralized view [9, 16]. Recently, two groups introduced distributed controllers for equal-mass partitioning ([65, 85]).

Our problem is related to the graph theory problem called *graph partitioning*, for finding subsets of a graph with equal node weights and minimum total weights of edges crossing between subsets. The problem is useful in applications including designing VLSI, efficient routing, parallel computation of finite element method (FEM), etc. Graph partitioning is also NP-hard [41], and there are many results from heuristic solutions [36]. Many distributed and geometry based solutions were proposed ([15, 71, 108, 83]). Our solution is unique since we specialized the problem by adding two constraints[1]:

[1]Our problem is also NP-hard and proof is omitted for space limit.

1. a node belongs to the nearest robot,

2. a robot can relocate itself only in its Voronoi region.

Many existing algorithms either arbitrarily assign a node to a partition (robot) or relocate centroids to any nodes. In this problem, it is not important that we obtain the obtaining minimum edge cut, since the cost of the edge cut does not affect the cost function that drives our controller. We focus on dividing a graph into subsets with equal node weights.

More formally, given the Voronoi partition V_i, we define its mass property as the sum of the target density function in the area.

$$M_{V_i} = \sum_{V_i} \phi_t(q). \tag{5.3}$$

If all the nodes have the same unit node-weight ϕ_t, then M_{V_i} is the number of nodes in V_i. The cost function is given by:

$$\mathcal{H}_E = \sum_{i=1}^{n} \frac{1}{M_{V_i}}. \tag{5.4}$$

Note that $\mathcal{H}_E$ is minimized only if $M_{V_1} = M_{V_2} = \ldots = M_{V_n}$

5.3.2 Control algorithm

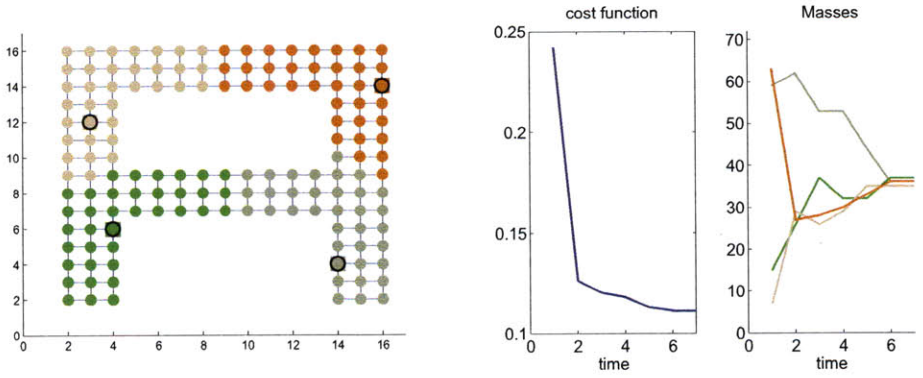
Algorithm 4 shows the distributed vertex substitution algorithm for equal-mass partitioning. The setup of the algorithm inherits Algorithm 3, and the local cost function $\mathcal{H}_{E_i}$ is defined as:

$$\mathcal{H}_{E_i} = \sum_{l=i, l \in \mathcal{N}_i} \frac{1}{M_{V_l}}. \tag{5.5}$$

Note that decay of $\mathcal{H}_{E_i}$ directly leads to the decay of the total cost function $\mathcal{H}_E$. As in locational optimization, we check how the Voronoi regions change and following change to the masses is ${}_j\Delta_l$ where $l = i, l \in \mathcal{N}_i$. The substitution to q_b that may lead to change $V_{\mathcal{N}_{\mathcal{N}_i}}$ is discarded as in Algorithm 3. The changed $\mathcal{H}_{E_i}$ by substituting p_i^* to q_b is denoted by $\hat{\mathcal{H}}_{E_i}$, and the optimal node for substitution is chosen so that it minimize $\hat{\mathcal{H}}_{E_i}$. The minimum should be smaller than $\mathcal{H}_{E_i}$, otherwise the algorithm returns null.

Algorithm 4 Distributed Vertex Substitution Algorithm for Equal-mass Partitioning

$D_i^Q = D(:, p_i^* \cup p_{\mathcal{N}_i}^* \cup p_{\mathcal{N}_{\mathcal{N}_i}}^*)$
$\mathcal{H}_{E_i} = \sum_{l=i, l\in\mathcal{N}_i} \frac{1}{M_{V_l}}$
for $q_b = \{q \in V_i | q \neq p_i\}$ **do**
  for $q_j = \{q \in V_i \cup V_{\mathcal{N}_i} \cup B_{\mathcal{N}_i} | q \neq p_{\mathcal{N}_i}\}$ **do**
    $d_{jk} \leftarrow \min(\text{row}(D_i^Q, j)$
    $r_k \leftarrow$ ID of the robot at k
    $d_{js} \leftarrow$ 2nd smallest row(D_i^Q, j)
    $r_s \leftarrow$ ID of the robot at s
    if $r_i = r_k$ **then**
      if $(d_{jb} = d_{js}$ and $r_i > r_s)$ or $d_{jb} > d_{js}$ **then**
        ${}_j\Delta_i \leftarrow_j \Delta_i - \phi_t(q_j)$
        ${}_j\Delta_s \leftarrow_j \Delta_s + \phi_t(q_j)$
      end if
    else
      if $d_{jb} < d_{jk}$ or ($d_{jb} = d_{jk}$ and $r_k > r_i$) **then**
        if $r_k \notin \mathcal{N}_i$ **then**
          discard q_b
        end if
        ${}_j\Delta_i \leftarrow_j \Delta_i + \phi_t(q_j)$
        ${}_j\Delta_k \leftarrow_j \Delta_k - \phi_t(q_j)$
      end if
    end if
  end for
  $\hat{\mathcal{H}}_{E_i}^b = \sum_{l=i, l\in\mathcal{N}_i} \frac{1}{M_{V_l} + \sum_j {}_j\Delta_l}$
end for
$\hat{\mathcal{H}}_{E_i} = \min \hat{\mathcal{H}}_{E_i}^b$
if $\hat{\mathcal{H}}_{E_i} \geq \mathcal{H}_{E_i}$ **then**
  return $\varnothing$
else
  return b^* of $\hat{\mathcal{H}}_{E_i}$
end if



(a) (b)

Figure 5-8: Simulation result from 4 robot coverage on the small bridge. (a) The final configuration of equal-mass partitioning on the small bridge. (b) The cost function $\mathcal{H}_E$ and the masses M_{V_i}

5.3.3 Analysis

Theorem 5 *Algorithm 4 converges to a local minimum.*

Proof: The proof has the same structure as the proof for locational optimization in Theorem 3. We replace $\mathcal{H}_L$ by $\mathcal{H}_E = \sum_{i=1}^{n} \frac{1}{M_{V_i}}$. □

5.3.4 Implementation

Algorithm 4 was implemented and tested on a suite of graphs including the graphs in Figures 5-3 and 5-5. Figure 5-8 shows the data for the small bridge in Figure 5-3. We see the masses converge to approximately the same value as in Figure 5-8(b). Compared to locational optimization (Figure 5-3), the locations of the robots look irregular since the robots do not have to be at the centroid. The resultant Voronoi regions from the distributed controller in a continuous domain is shown in Figure 5-9 ([65]), and they look similar as well. Algorithm 4 guarantees that V_i is fully connected while the distributed controller in [65] does not.

Figure 5-10(a) shows the final Voronoi regions by the equal-mass partitioning controller with 15 robots on the big bridge. The masses converge as shown in Figure 5-10(b).

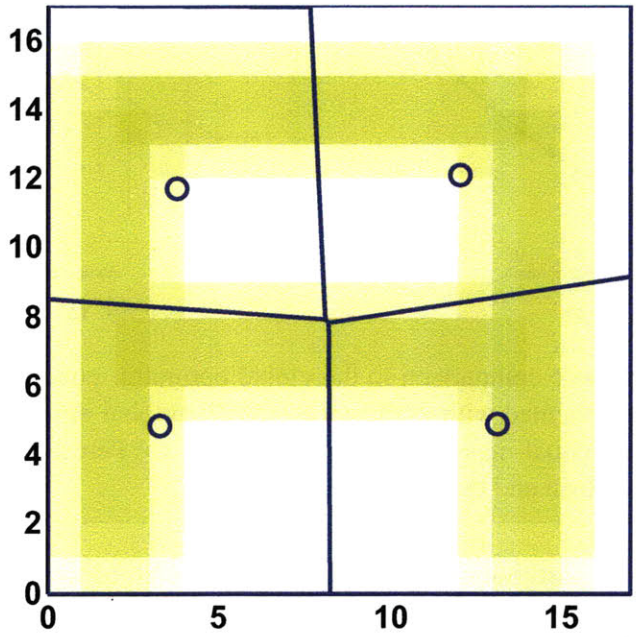
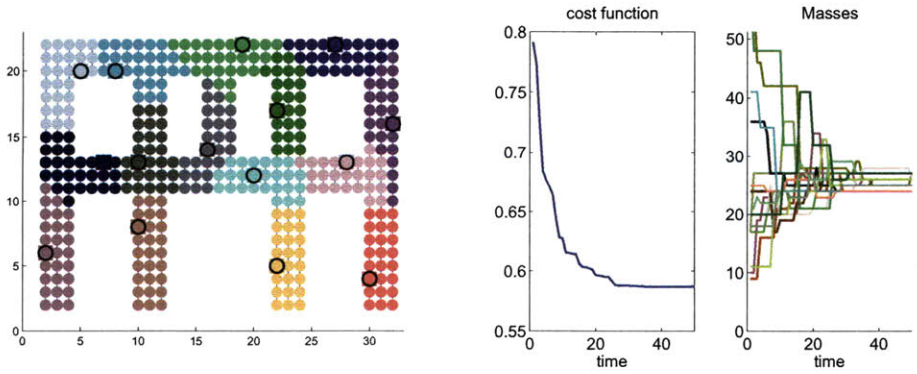
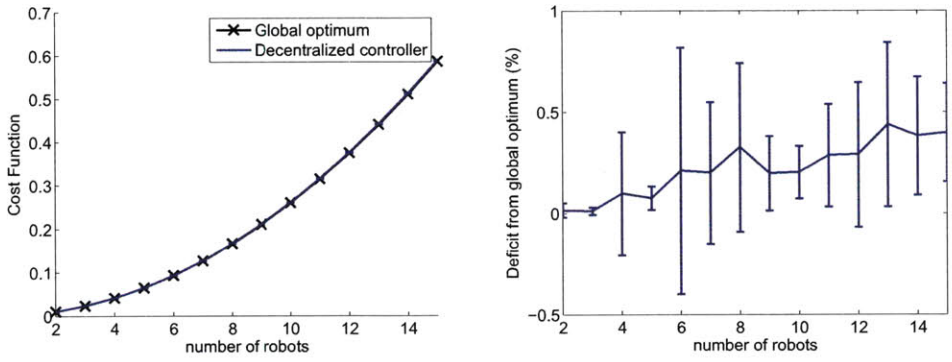


Figure 5-9: The resultant Voronoi regions by equal-mass partitioning with the continous density function. The distributed controller proposed in [65] is used.



(a) (b)

Figure 5-10: Simulation result from 15 robot coverage on the small bridge. (a) The final configuration of equal-mass partitioning on the small bridge. (b) The cost function $\mathcal{H}_E$ and the masses M_{V_i}.



(a) (b)

Figure 5-11: Performance comparison to the global optimum. equal-mass partitioning on the big bridge is implemented with 2-15 robots. (a) The global optimum and the resultant cost function from the equal-mass partitioning controller (b) Percentage of deviation from the global optimum. Mean and error-bars are shown.

For the equal-mass partitioning problem, we know the global optimum of $\mathcal{H}_E$:

$$\mathcal{H}_{E_{opt}} = \frac{n^2}{M_G}, \tag{5.6}$$

where M_G is the total mass of G. Therefore we can compare the result from our controller to $\mathcal{H}_{E_{opt}}$ as in Figure 5-11. The two plots for the global optimum and for our controller are almost identical. Figure 5-11(b) confirms that the deviation from the global optimum is less than 1% independent of how many robots were used in the test.

Chapter 6

Delivery and Assembly Algorithms

Once the assembling robots are in place according to the equal-mass partitioning controller, construction may begin. State machines drive the delivering robots and the assembling robots. During construction we wish to distribute the source components (truss elements and connectors) to the assembling robots in a balanced way. Global balance is asymptotically achieved by a probabilistic target selection of delivering robots that uses ϕ_t as a probability density function. For local balance, the delivering robots are driven by the gradient of demanding mass defined as the remaining structure to be assembled by the robot. Robots with more work left to do get parts before robots with less work left. Each assembling robot waits for a new truss element or connector and assembles it to the most demanding location in *its Voronoi region*. Therefore, construction is purely driven by the density functions regardless of the amount of the source components and it can be done without an explicit drawing of the target structure. We ensure that all the processes of the controllers work in a distributed way and each robot needs to communicate only with neighbors. Details of the control algorithms are explained next.

6.1 Probabilistic Delivery with Local Gradient Search

delivering robots operate by a state machine as shown in Figure 6-1. Each robot has the following states:

- IDLE

- ToSOURCE: moving to get a new element
- ToTARGET: moving to a picked point at the target area Q
- ToASSEMBLY: delivering the element to an assembling robot

Algorithm 5 describes the details of the state machine.[1] Given an initially empty state, a delivering robot changes its state to ToSOURCE and moves to $\mathcal{S}$ (the source location). At $\mathcal{S}$, the robot picks a source component if one exists. Otherwise, it stops working. The state is switched to ToTARGET and the robot moves to a randomly chosen point in Q following the probability density function ϕ_t. Therefore, materials are more likely to be delivered to an area with a denser ϕ_t. After arrival at the chosen point, the robot changes the state to ToASSEMBLY and moves following the gradient of the demanding mass ΔM_{V_i} of assembling robots. Delivery by the gradient of the demanding mass yields a *locally* balanced mass distribution. Note that the global balance is maintained by the randomly chosen delivery with density ϕ_t. When the robot meets the assembling robot with the maximum demanding mass, it checks if the state of the assembling robot is WAITING and passes the material. The state changes to ToSOURCE and the robot repeats delivery.

6.2 Greedy Assembly Algorithm

Each assembling robot operates using a state machine as shown in Figure 6-2. The robot has the following states:

- IDLE
- WAITING: waiting for a new component
- MOVING: moving to the optimal location to add the part
- ASSEMBLING: adding the component to the assembly

[1]The assembly and the delivery algorithms provably guarantee completion of the correct target structure. In the interest of space, the proof is omitted. Empirical results in Section 6.3 shows correctness of the algorithms since all the simulations with different initial conditions end up with the same final structure.

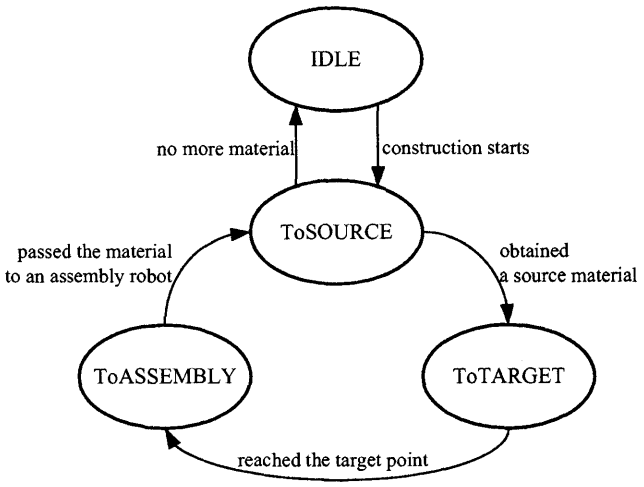


Figure 6-1: The state machine for a delivering robot. A delivering robot repeatedly passes source components from the source location to an assembling robot. The initialization of construction causes the delivering robots to start moving. The robots finish working when there is no more source material left at the source location or the assembly is complete.

Algorithm 5 Control Algorithm of delivering robots

STATE: IDLE
1: *state* = ToSOURCE
2: $\mathbf{t} = \mathcal{S}$
STATE: ToSOURCE
3: **if** reached $\mathbf{t}$ **then**
4: **if** source material remains **then**
5: pick a material element
6: $\mathbf{t} = \mathbf{q}, \mathbf{q} \sim \phi_t(\mathbf{q})$
7: *state* = ToTARGET
8: **else**
9: *state* = IDLE
10: **end if**
11: **else**
12: move to $\mathbf{t}$
13: **end if**
STATE: ToTARGET
14: **if** reached $\mathbf{t}$ **then**
15: *state*=ToASSEMBLY
16: **else**
17: move to $\mathbf{t}$
18: **end if**
STATE: ToASSEMBLY
19: communicate with robot r_i s.t. $\mathbf{q} \in V_i$
20: $deliveryID = \text{argmax}_{(k=i,j\in\mathcal{N}_i)} \Delta M_{V_k}$
21: $\mathbf{t} = p_{deliveryID}$
22: **if** reached $\mathbf{t}$ & *state* of r_i = WAITING **then**
23: pass the material
24: *state* = ToSOURCE
25: $\mathbf{t} = \mathcal{S}$
26: **else**
27: move to $\mathbf{t}$
28: **end if**

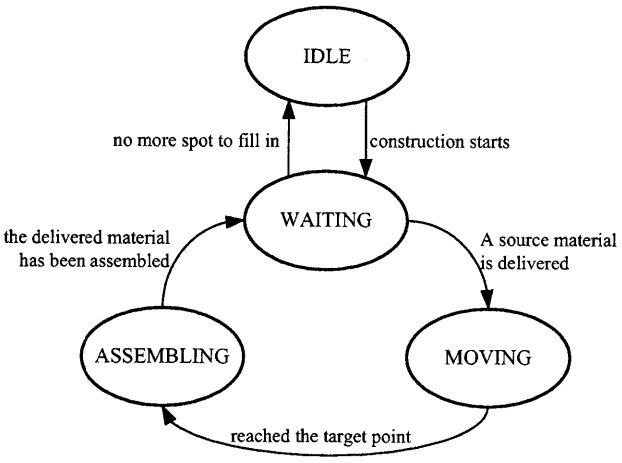


Figure 6-2: The state machine for an assembling robot. Each assembling robot waits for the delivery of a source component, moves the component to the optimal spot and adds it to the structure. The robot's task is complete when there is no demanding mass left.

Each robot has a graph representation $G_i = (R_i, E_i)$ of the already built substructure. The graph is composed of sets of nodes and edges in the Voronoi region. For simplicity of exposition, we assume truss elements of two sizes: the unit-box size, and the unit box diagonal. The extension to multiple sizes is trivial. We design the density function according to a grid. The unit length of the grid is the length of the truss element. Vertices of the grid have density values equal to the number of truss elements at the vertex. The density of the intermediate points in the space is interpolated. The interpolated value is used in the coverage implementation only. We can generalize this cost function to be a continuous function that encodes the geometry of the object. The demanding mass is defined uniquely for each component type. As for a truss element, the demanding mass $\Delta M^t_{V_i}$ is computed as:

$$\Delta M^t_{V_i} = \int_{V_i} \phi_t(\mathbf{q}) d\mathbf{q} - \int_{V_i} \rho(\mathbf{q}) d\mathbf{q}, \tag{6.1}$$

where $\rho(\mathbf{q})$ is the density function of the built structure, which increases as a robot assembles truss elements. Note $\phi_t(\mathbf{q})$ of the target shape is fixed. Therefore, a bigger demanding mass means that more elements should be included in that area. The demanding mass for connectors $\Delta M^c_{V_i}$ is the number of required connectors Φ^c for the current structure G_i.

Note that $\Delta M^c_{V_i}$ is a function of $\phi(\mathbf{q})$. The demanding masses drive a delivering robot according to gradients as in (Section 6.1). If a structure is composed of other components, we can define the demanding mass for each material.

Algorithm 6 shows the details of the state machine. When construction starts, an assembling robot initializes the parameters R, E, ρ, Φ^c and changes its state to WAITING. Once a new truss element is delivered, the robot finds the optimal place to add it to the structure using Algorithm 7. Since we want the structure to gradually grow, the optimal edge is chosen among a set of edges E_1 that are connected to G. Let E_2 be a set of edges that have maximum demanding mass in E_1. The demanding mass of an edge can be computed as the sum of masses of two nodes defining the edge. Each node of the edges in E_2 should have a density value greater than the threshold preventing the robot from assembling the component outside the target structure. In order to achieve a spreading-out structure, priority is given to unconnected edges. If no such edge exists, we choose another seed edge that is not connected to G and has the maximum demanding mass. This jump is required in case that the robot covers substructures which are not connected to each other. If the delivered material is a connector, the optimal location is a node $v \in \Phi^c$ that is connected to the largest number of edges in E. The state machine sets a target location $\mathbf{t}$ according to the optimal location and changes the state to MOVING. In the MOVING state, an assembling robot moves to the target location $\mathbf{t}$ and changes the state to ASSEMBLING when it arrives. Finally, a robot assembles the delivered material and updates the parameters. It adds a node of the optimal edge to Φ^c if the node $\notin \Phi^c$ and is connected to other edges. If the material is a connector, the robot removes the node from Φ^c. The state switches to WAITING again.

6.3 Implementation

Figure 6-3 shows snapshots from the simulation after partitioning. We use 4 robots for truss delivery and 4 robots for connector delivery. They deliver source materials which have 250 side truss elements and 150 connectors. The area with high density is gradually filled with truss elements and connectors. Because the controller uses equal mass partitioning and the

Algorithm 6 Control Algorithm of assembling robots

STATE: IDLE
1: $R = \varnothing, E = \varnothing$
2: $\rho(\mathbf{q}) = 0, \Phi^c = \varnothing$
3: *state*=WAITING
STATE: WAITING
4: **if** truss delivered **then**
5: e=findOptimalEdge(R, E, ϕ_t, ρ) (Alg. 7)
6: **if** $e \neq \varnothing$ **then**
7: $\mathbf{t} = \mathbf{q}_{(node_1(e)+node_2(e))/2}$
8: *state*=MOVING
9: **else**
10: *state*=IDLE
11: **end if**
12: **end if**
13: **if** connector delivered **then**
14: $v \leftarrow \Phi^c$
15: $\mathbf{t} = \mathbf{q}_v$
16: *state*=MOVING
17: **end if**
STATE: MOVING
18: **if** reached $\mathbf{t}$ **then**
19: *state*=ASSEMBLING
20: **else**
21: move to $\mathbf{t}$
22: **end if**
STATE: ASSEMBLING
23: assemble the material
24: **if** the material = truss **then**
25: update $\rho(e)$
26: **if** $node_2 \in R$ and $node_i \notin \Phi^c$ **then**
27: $\Phi^c \leftarrow node_i$
28: **end if**
29: $E \leftarrow e$
30: $R \leftarrow \{node_1(e), node_2(e)\}$
31: **end if**
32: **if** the material = connector **then**
33: $\Phi^c = \Phi^c - \{v\}$
34: **end if**
35: *state*=WAITING

Algorithm 7 Finding the Optimal Edge to Build

1: $E_1 = \varnothing, E_2 = \varnothing, E_3 = \varnothing$
2: **if** $E_1 = \varnothing$ **then**
3: $e_{opt} = \text{argmax}_e(\phi_t(e) - \rho(e)) \cap (\phi_t(e) > \Delta\phi_{threshold})$
4: **else**
5: $E_1 \leftarrow e, (e \notin E, node(e) \in R)$
6: $E_2 \leftarrow \text{argmax}_{e \in E_1}(\phi_t(e) - \rho(e)) \cap (\phi_t(e) > \Delta\phi_{threshold})$
7: **if** $E_2 = \varnothing$ **then**
8: $e_{opt} = \text{argmax}_e(\phi_t(e) - \rho(e)) \cap (\phi_t(e) > \Delta\phi_{threshold})$
9: **else**
10: $E_3 \leftarrow e, (e \in E_2, \{node_1(e), node_2(e)\} \in \{R_i, R_{j,j \in \mathcal{N}_i}\})$
11: **if** $E_3 \neq E_2$ **then**
12: e_{opt}= random($E_2 - E_3$)
13: **else**
14: e_{opt}= random(E_2)
15: **end if**
16: **end if**
17: **end if**
18: **return** e_{opt}

gradient of the demanding mass, the assembling robots maintain almost the same ΔM_V all the time. Therefore, each Voronoi region has a balanced amount of truss elements. Note that the control algorithms do *not* depend on the amount of the source truss elements. With fewer elements, we obtain a thinner structure, while the availability of more truss element yields a denser structure. At the end of the simulation, the assembling robot that has built the least amount of the truss component has assembled 58 truss elements while the robot with the maximum amount has assembled 63. The robot with the minimum number of connectors assembled 33 connectors and the robot with the maximum number assembled 38.

Figure 6-4 shows the demanding masses for a truss part and a connector. All four curves are completely overlapped, meaning all the substructures have been balanced at all time. The demanding mass for a connector oscillates since it depends on the already built substructure.

6.3.1 Constructing an Airplane

Figure 6-5 and Figure 6-6 shows snapshots of building a 3D pyramid and an airplane. 3D grids are used and the target density functions are given and computed in the grids.

Figure 6-6 shows snapshots of building an airplane. 3D grids are used and the target density functions are given and computed in the grids.

6.4 Analysis of the Algorithms

We now build on the algorithms and analyze the performance of the algorithms with respect to balance among the substructures and completion time. Simulation data is obtained from building the A-shaped bridge in Figure 4-4.

6.4.1 Balance of the sub-structures

Our goal is an algorithm that ensures the subassembly tasks proceed and get completed at the same time. This ensures that the overall construction is well-parallelized and there

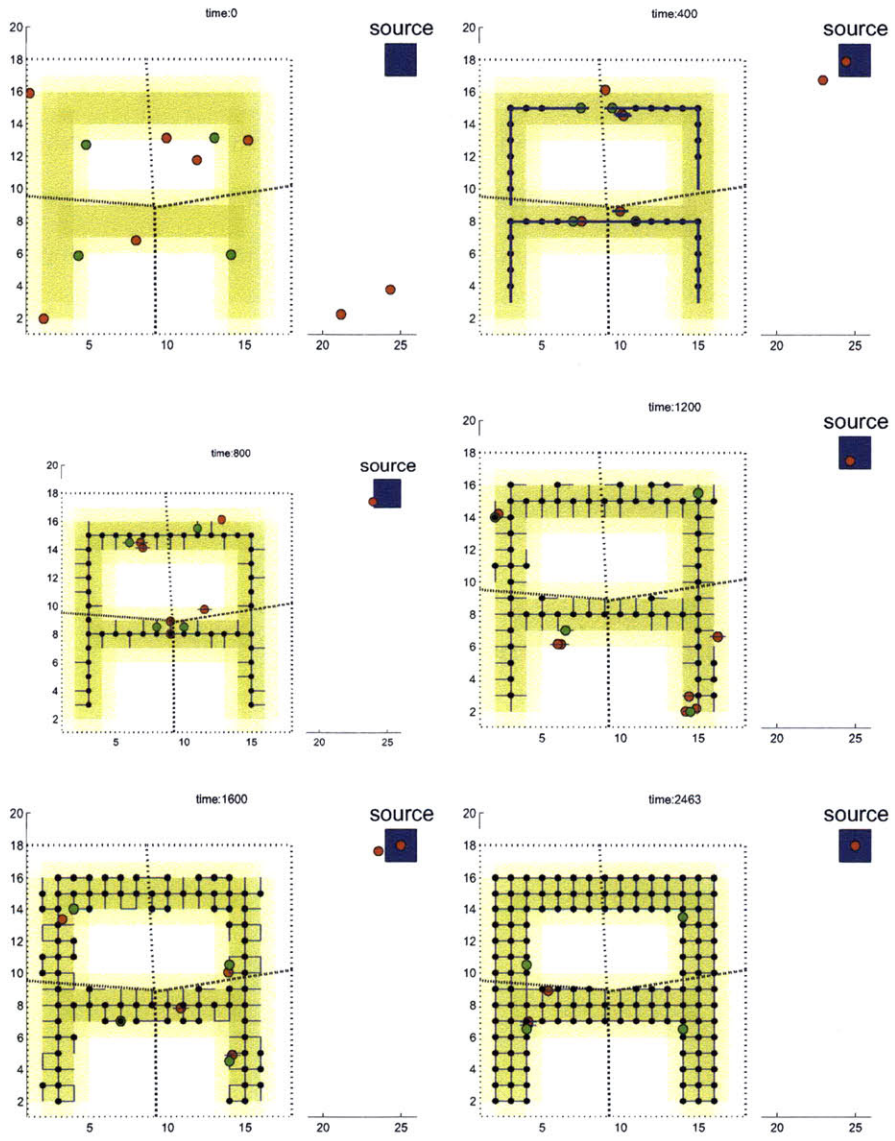
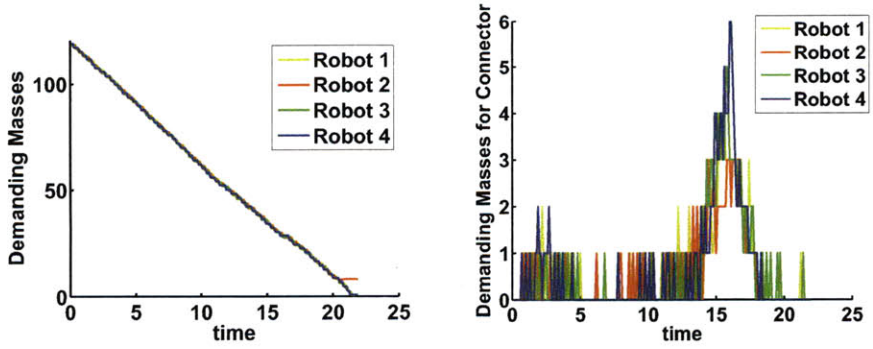


Figure 6-3: Snapshots of simulation. Green circles denote assembling robots and red circles denote delivering robots. The blue line is a truss elements and the black dot is a connector. The blue box is the source location. The dotted lines in Q are boundaries of the Voronoi regions.



(a) (b)

Figure 6-4: (a) Demanding masses for a truss part and (b) a connector. 4 assembling robots and 8 delivering robots are used. The assembly time is set to ten times the velocity. All the graphs are almost overlapped.

is no unnecessary waiting for subassembly completion. Let us assume the equal-mass partitioning is successful so that each assembling robot has the same amount of the target structure. The probabilistic deployment of the delivery algorithm leads to the traditional problem *ball-into-bins* where we throw m balls into n bins one by one with uniformly distributed probability of placing a ball at a bin. This problem is also known as *online load balancing* for distributed computation, where n servers are supposed to match m requests. In both cases, the question is what is the maximum number of balls (requests) in any bin (server).

Theorem 6 *With only probabilistic deployment, the maximum deviation of delivery from the mean ($\frac{m}{n}$) is bounded by $\sqrt{2\frac{m}{n}\log n}$ with high probability.*

Proof: In case $m \gg n$ as ours, with high probability (normally $\gg 1 - \frac{1}{n}$), the maximum number of balls [90] is smaller than

$$\frac{m}{n} + \sqrt{2\frac{m}{n}\log n}. \tag{6.2}$$

Since the mean number of balls is $\frac{m}{n}$, The maximum deviation from the mean is bounded by $\sqrt{2\frac{m}{n}\log n}$. □

Figure 6-7(a) shows the demanding masses simulated from an example where 10 assembling robots and 10 delivery robots are used and only the probabilistic deployment

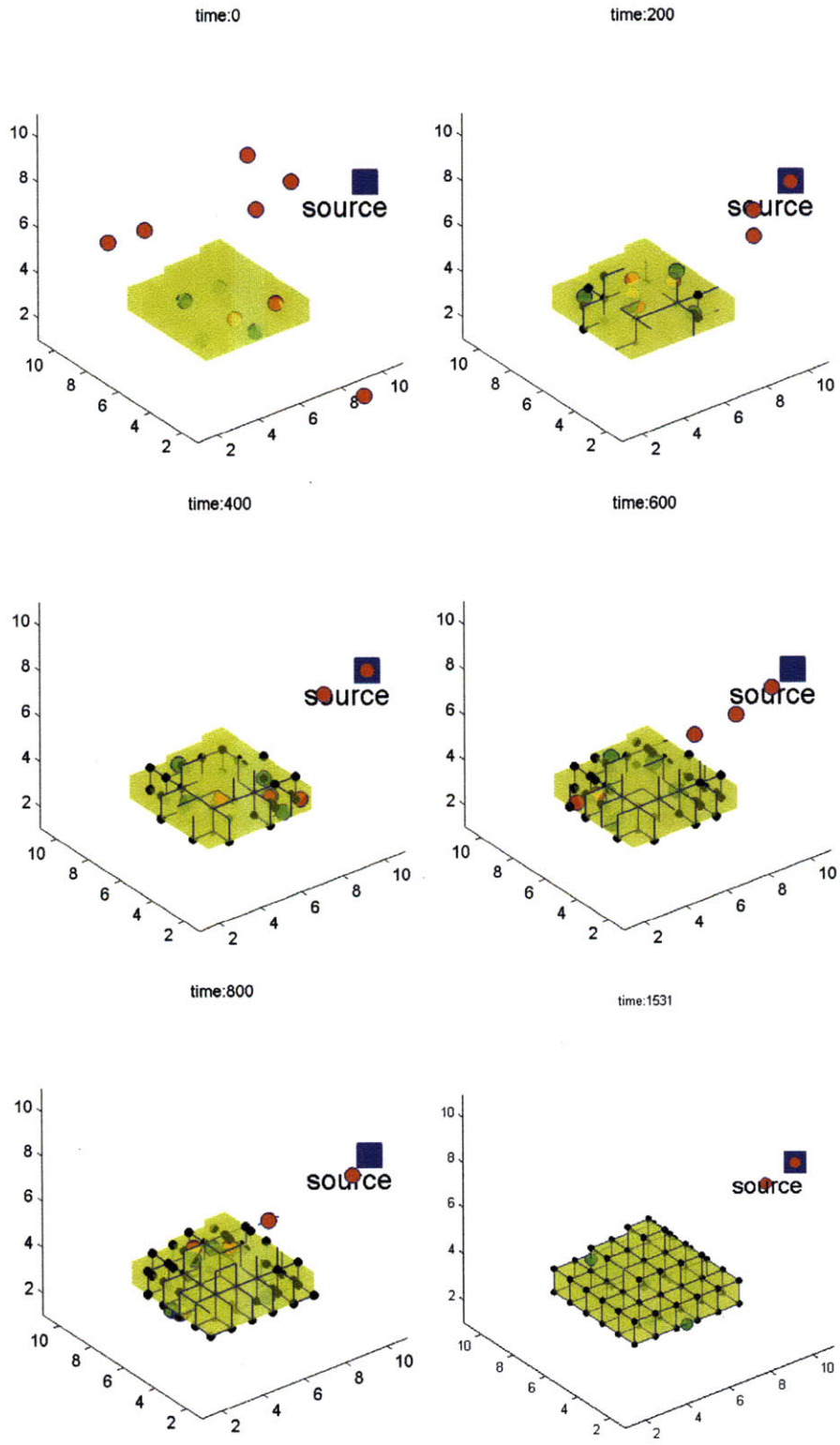


Figure 6-5: Snapshots of building a pyramid. 4 assembling robots and 8 delivering robots work. 189 side truss elements are used.

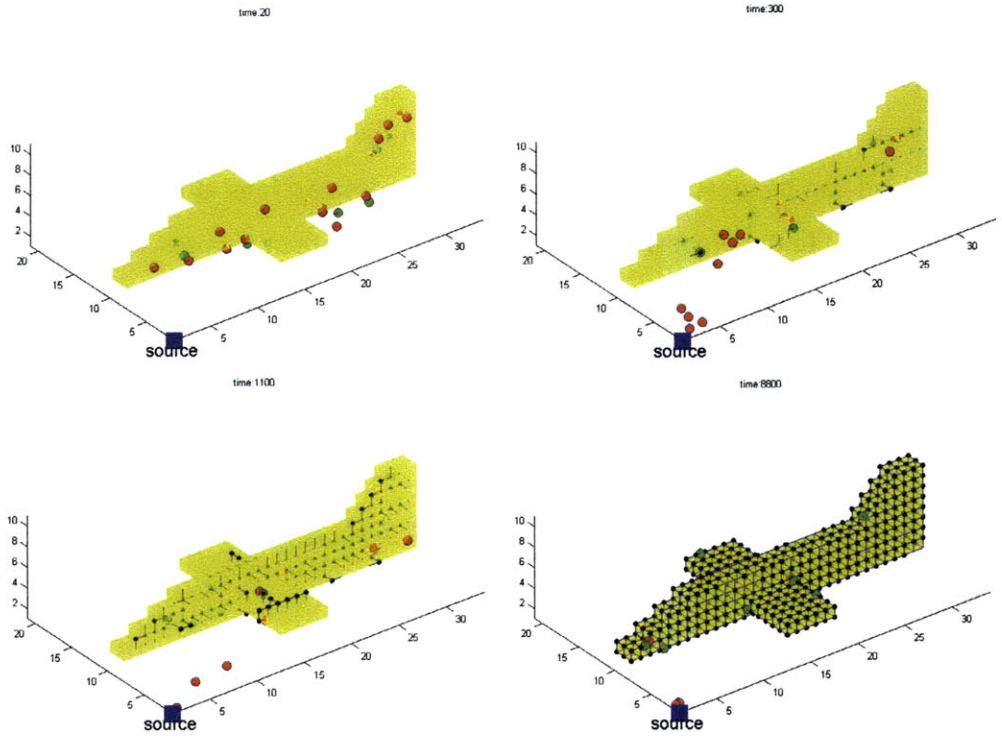


Figure 6-6: Snapshots of building an airplane. There are 10 assembling robots, 20 delivering robots for truss parts and 10 robots for connectors. 1575 truss parts and 656 connectors are assembled.

is implemented. We can see the demanding masses spread out as construction goes on. Figure 6-8 shows maximum deviation of the demanding mass from the mean and the theoretical bound. The mean of the maximum deviation and the error bars are obtained from 10 simulations.

Algorithm 1 allows a delivering robot to find the assembling robot with the maximum demanding mass after the probabilistic deployment, and that dramatically improves balance as shown in Figure 6-7(b) and Figure 6-8. During construction, all the demanding masses are within a range of a single truss element, which implies perfect balance. This local search can be understood as picking multiple bins first and putting a ball at the bin with the minimum number of balls. It is well known in the balls into bins problem that the maximum load can be greatly reduced if we can choose two bins at random rather than just one bin [80]. In the proposed algorithm, a delivering robot chooses where to place a source component among neighboring robots of the robot that is picked by the probabilistic deployment. This is equivalent to having the robot choose multiple assembling robots on a graph.

Theorem 7 *Algorithm 1 yields the maximum deviation bounded by* $\frac{\log\log n}{\log 2}$ *with high probability.*

Proof: The maximum load decreases into [58]

$$\frac{\log\log n}{\log d} + \frac{m}{n}, \tag{6.3}$$

where d is a number of bins we can choose. [2] Since we do not know how many neighbor robots there are, we use a conservative bound with $d = 2$. □

The black dotted line is the bound with $\frac{\log\log n}{\log 2}$. Note that the maximum deviation is not dependent on m.

[2] To qualify the equation, the graph should be regular with degree n^{ϵ} where ϵ is not too small [58]. In our case, we can not guarantee a degree of the graph that equal-mass partitioning would build. However, if the target structure is fully connected, ϵ should be at least greater than 2.

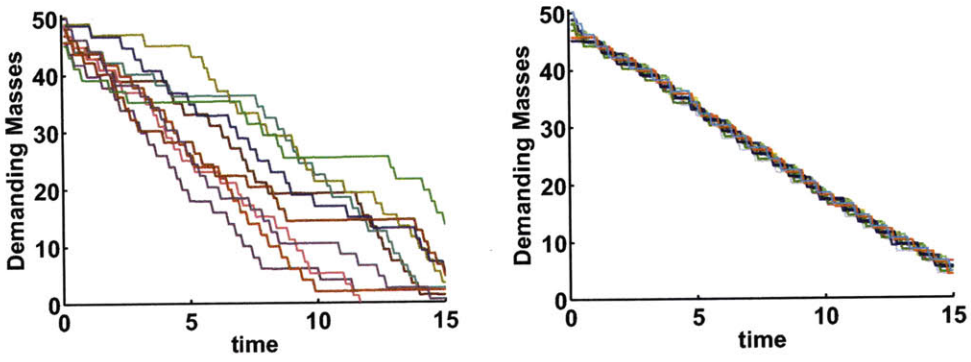


Figure 6-7: Demanding masses resulted from (a) probabilistic deployment only (b) proposed algorithm. 10 assembling robots and 10 delivering robots are used.

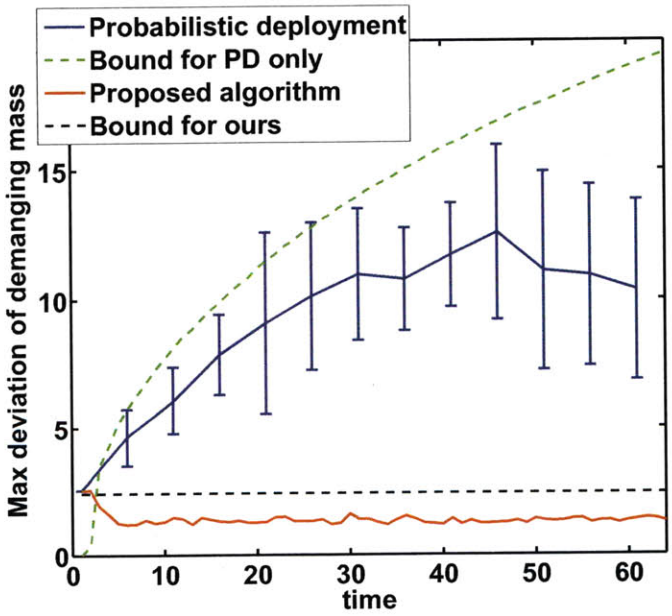


Figure 6-8: Average demanding mass and error bars from the probabilistic deployment and Algorithm 1. Green dotted line is the theoretical bound when only probabilistic deployment is used for delivery. Red solid line is the simulation result from the proposed algorithm, and black dotted line is the theoretical bound for the algorithm. The bound makes more sense when enough time has passed, since the bound is valid for $m \gg n$.

6.4.2 Construction time and Travel distance

We conduct an empirical analysis of the construction algorithms, by testing several combinations of parameters. There are two major parameters that affect the total construction time: velocity of the robot and assembly time required for an assembling robot to assembling a part. If the assembly time is much larger than the reciprocal of the velocity, construction time will be dominated by the assembly time. If the assembly time is very short, the total time will be a function of the traveling distances of the robots. We evaluate the algorithms with the following sets of parameters: $N_a \in \{1, 2, 4, 10\}$, $N_d/N_a \in \{1, 2, 4\}$, $T_a \in \{1, 5, 20\}$. T_a is the assembly time.

When the assembly time is large, the construction time decreases proportional to the number of the assembly robots, as shown in Figure 6-9(a). Therefore the control algorithms yield good parallelism when a robot has a large assembly time. If the assembly time is small, we may modify the criteria for a delivery robot to select an assembly robot by incorporating expected traveling distance. This will be considered in our future work.

The average travel distance of the delivery robots is examined in Figure 6-9(b). Increasing the number of delivery robots is more effective when the number of assembly robots is small. However, too many delivery robots do not reduce the average distance and the construction time much (the slopes become flat as the number increases.) Careful choice of the robot numbers will yield the an appropriate tradeoff between robot numbers and construction time. This will be investigated in the future.

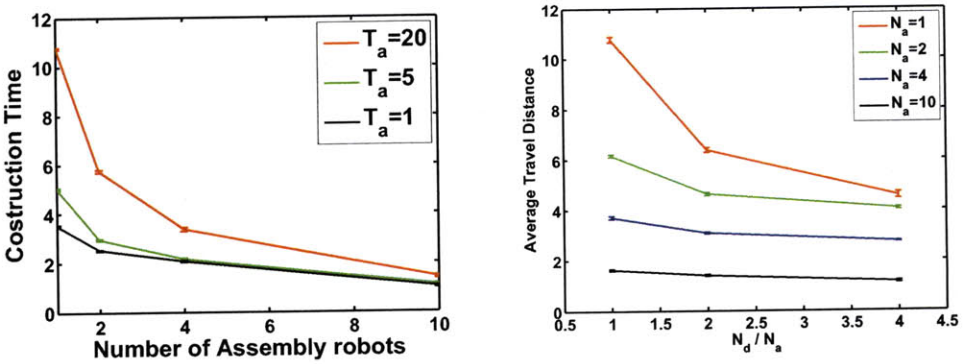


Figure 6-9: (a)Total construction time. (b)Average travel distance of the delivery robots. Error bars are plotted together. The construction time is down-scaled by 1000 for a better view.

Chapter 7

Adaptation in Construction

The construction algorithms in Section 6.4 are adaptive to several cases such as failure of robots, construction with dynamic constraints, multiple types of source elements and reconfiguration between two structures. We next discuss each case.

7.1 Dynamic constraint: Construction in Order

Territorial construction is subject to gravity constraints which in turn imposes ordering constructions on assembly job. For example, a 3D structure should be built from the ground up. We extend our algorithm to incorporate this type of constraint in terms of connectivity. Given ϕ_t, we ensure connectivity by revealing only the part of ϕ_t that is connected to the current structure. Equal-mass partitioning and the computation of the demanding mass are done with the revealed part of ϕ_t, which is now a time-varying function. We model this revealed part of ϕ_t as a time-varying target density function φ_t. The assembling robots perform equal-mass partitioning based on φ_t.

We update φ_t by Algorithm 8. Given the grid map Q, R_c is a set of nodes that are reachable, Φ_0 is a unit density for each node of a truss element, and T_a is an assembly time to finish assembling a truss element. When an assembling robot starts to build a truss element at an edge e_{opt}, it checks whether the adjacent nodes of e_{opt} are in R_c or not. For the nodes to be revealed $\mathbf{q}_1$, the density function increases by the rate $\frac{\phi_t}{T_a}$ till time T_a. Therefore, only the nodes connected to the current structure (R_c) are used in the current target density

Algorithm 8 Update the density function φ_t during building a single truss element

1: $\mathbf{q}_1 \leftarrow$ a set of nodes incident to e_{opt} and $\notin R_c$
2: $\mathbf{q}_2 \leftarrow$ two nodes of e_{opt}
3: set $t = 0$
4: **repeat**
5: $\quad \dot{\varphi}_t(\mathbf{q}_1) = \frac{\phi_t(\mathbf{q}_1)}{T_a}$
6: $\quad \dot{\rho}(\mathbf{q}_2) = \frac{\Phi_0(\mathbf{q}_1)}{T_a}$
7: $\quad$ update φ_t and ρ
8: **until** $t > T_a$
9: $R_c \leftarrow \mathbf{q}_1$

function φ_t. The next chosen edge e_{opt} must be connected to the current structure.

The coverage control follows Algorithm 9. We modify it to incorporate the time varying density function. Note that φ_t varies smoothly since $\dot{\varphi}_t$ is a constant.

Given the cost function $\mathcal{H}$ that is now a function of φ_t replacing ϕ_t in Equation 4.5, differentiating $\mathcal{H}$ yields

$$\dot{\mathcal{H}} = \sum_{i=1}^{n} \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}_i} \dot{\mathbf{p}}_i + F_i \prod_{k \notin \{i, \mathcal{N}_i\}} M_{V_k} \right). \tag{7.1}$$

The new term F_i comes from the time varying density function, and can be computed as

$$F_i = - \prod_{j=\mathcal{N}_i} M_{V_j} \int_{V_i} \dot{\varphi}_t(\mathbf{q}, t) d\mathbf{q}, \tag{7.2}$$

where $\dot{\varphi}_t$ is given by Algorithm 8. If we set the velocity input as

$$\dot{\mathbf{p}}_i = \frac{\boldsymbol{J}_i}{\|\boldsymbol{J}_i\|^2 + \lambda^2} (k - F_i) \tag{7.3}$$

where

$$\boldsymbol{J}_i = \sum_{j=i,\mathcal{N}_i} \frac{\partial M_{V_j}}{\partial \mathbf{p}_i} \prod_{k \in \{i,\mathcal{N}_i\}, k \neq j}^{n} M_{V_k}, \tag{7.4}$$

$\dot{\mathcal{H}}$ becomes

$$\dot{\mathcal{H}} = - \sum_{i=1}^{n} \frac{1}{\|\boldsymbol{J}_i\|^2 + \lambda^2} \left(k \|\boldsymbol{J}_i\|^2 + \lambda^2 F_i \right) \prod_{l \notin \{i,\mathcal{N}_i\}} M_{V_l}. \tag{7.5}$$

Theoretically, setting the gain k to a large value ensures $\dot{\mathcal{H}} \leq 0$ unless all $\boldsymbol{J}_i$ are zero.

We conjecture that F_i also becomes zero if all $\boldsymbol{J}_i$ are zero, however, we have not proven this yet. In practice, a robot sets the gain k_i that guarantees a local derivative of the cost function $\dot{\mathcal{H}}_i \leq 0$, which is defined as

$$\dot{\mathcal{H}}_i = \sum_{j=i, j\in\mathcal{N}_i} \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}_j} \dot{\mathbf{p}}_j + F_j \prod_{k\notin\{i,\mathcal{N}_i\}} M_{V_k} \right). \tag{7.6}$$

Theorem 8 *F_i is bounded.*

Proof: M_V and $\dot{\varphi}_t$ are bounded. Therefore, by Equation (7.2), F_i is bounded. □

We can show that F_i can be more tightly bounded in a discrete domain if we assume that all the robots have equal mass before the change of the density function.

$$|F_i| \leq \sum_{j\in\mathcal{N}_i} \left(M_{V_i}(t_-) - M_{V_j}(t_-) + 2\left|\dot{\varphi}_t\right|_{max} \Delta t \right) 2\left|\dot{\varphi}_t\right|_{max} \tag{7.7}$$

$$\approx 4\left|\mathcal{N}_i\right| \left|\dot{\varphi}_t\right|^2_{max} \Delta t \tag{7.8}$$

where $M_{V_i}(t_-)$ is the mass before change, $\left|\dot{\varphi}_t\right|_{max}$ is the maximum changing rate of the density function, Δt is a control sampling time, and $|\mathcal{N}_i|$ is the number of neighbors.

Theorem 9 *The control input $\dot{\mathbf{p}}_i$ is bounded.*

Proof: Because F_i and M_V are bounded, $\dot{\mathbf{p}}_i$ is bounded by Equation (7.3) □

Figure 7-1 shows results from our implementation of the control algorithms with 2 assembling robots. The bridge is to be built from the lower left corner. Only the lower left part of the target density function is revealed as in Figure 7-1(a). The more the robots build, the more of ϕ_t is used until the entire target density function ϕ_t is revealed. As shown in Figure 7-2(a), the cost function is almost flat even though φ_t changes during construction, since the controller incorporate the time varying density function. We can see the masses of two robots are almost identical at all time during construction as in Figure 7-2(b).

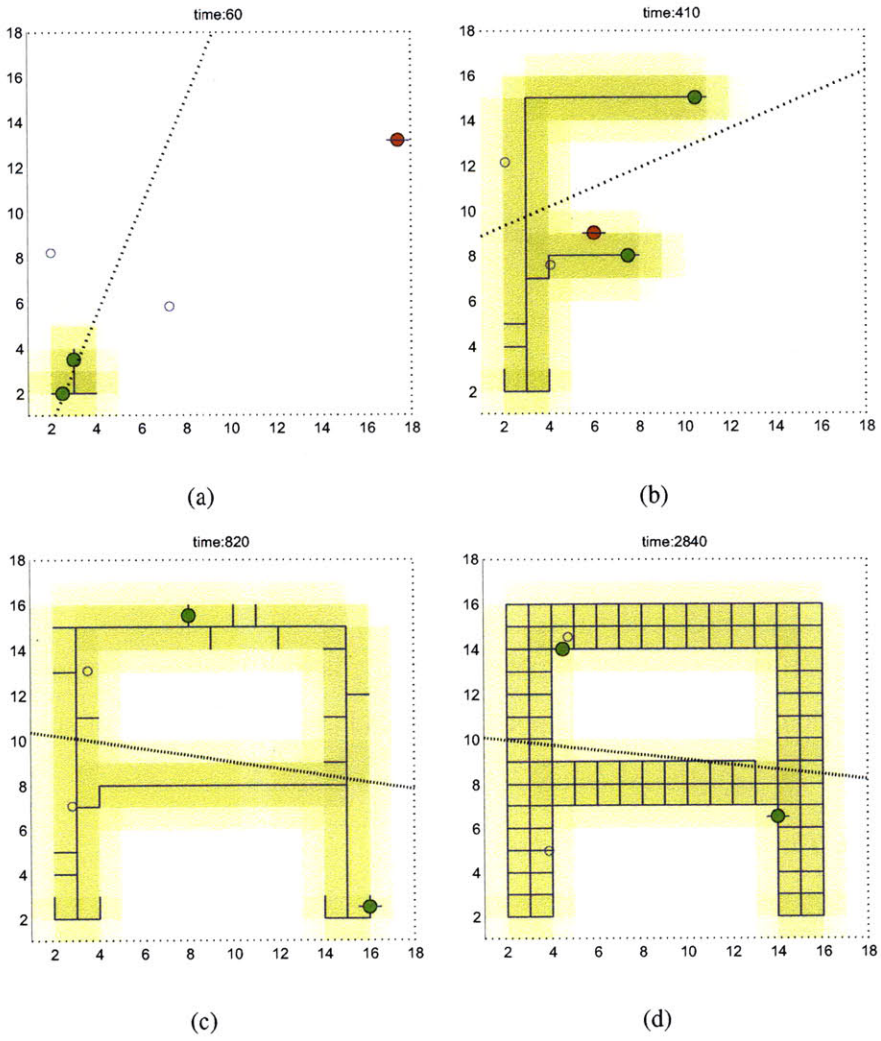
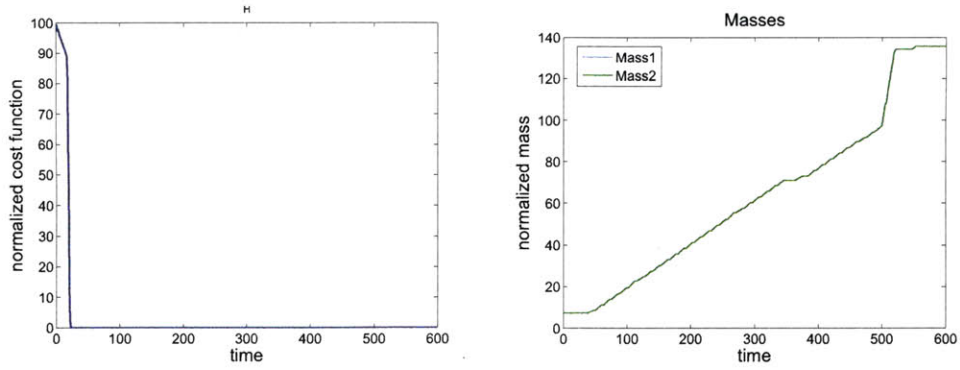


Figure 7-1: Construction in order. 2 assembling robots and 2 delivering robots are building the bridge from the lower left corner.



(a) (b)

Figure 7-2: Construction in order

7.2 Robustness to Robot Failure

Assembling robots are critical since each assembling robot covers a unique region. Failures of the assembling robots can be tolerated by executing the subassembly equal-mass partitioning continuously as a background process. When a robot fails, its remaining subassembly task will get reassigned and all the other assembly loads re-balanced. We assume a failed robot disappears with an element if it is carrying any. Algorithm 9 shows the main

Algorithm 9 Assembly with Equal-mass Partitioning

1: **repeat**
2: assemble the delivered components
3: move to $\hat{\mathbf{p}}_i$ by Equation 4.16
4: update V_i, G, ρ, Φ^c
5: **until** task completed

control loop for assembling robots with continuous equal-mass partitioning. ρ describes a density function for currently built structure, and Φ^c is a set of required connectors for the current structure. The assembling robots reconstruct the Voronoi regions when the surrounding network of the robots has changed. Since assembling robots move during construction, we introduce the virtual center of the Voronoi region $\hat{\mathbf{p}}_i$ and move it instead of a robot position, and reconstruct V_i around $\hat{\mathbf{p}}_i$. The assembling robots also need to update the parameters such as the graph of the built structure and demanding mass for truss and connectors. We assume that a robot can detect failure of its neighbor.

Theorem 10 *Continuous coverage during construction compensates for the failure of the assembling robots*

Proof: The coverage controller guarantees decay of the cost function $\mathcal{H}$ regardless of the number of neighbors. Therefore, if a robot fails, $\mathcal{H}$ will decrease to a local optimum with the changed configuration, as long as there are the remaining assembling robots. □

Figure 7-3 shows a snapshot from a simulation with a failed robot. The robot in the upper right Voronoi region fails during construction as Figure 7-3(b), and the neighboring robots adapt their Voronoi regions to fill the region of the failed robot while continuing construction. Since the coverage control requires a significant amount of computation, the robots end it when the cost function settles down as shown in Figure 7-4.

Failure of delivering robots is not critical in our approach, because the system is transparent to that. Only the completion time would increase, since we have less number of delivering robots after the failure.

7.3 Reconfiguration

The goal structure might change after or during construction. We extend the construction algorithm to support adaptation to changing structure geometry during construction, in order to build a new goal structure from the current structure. Suppose a target structure ϕ_{t_1} has been built and a new target structure ϕ_{t_2} is given. Assuming the assembling robot is capable of disassembly, Algorithm 10 shows how the original structure is reconfigured to the new structure. Here we set the target density function as difference between two structures $|\phi_{t_2} - \phi_{t_1}|$ for equal-mass partitioning, since disassembly also requires work of assembling robots. We assume cost for disassembly is the same as assembly. If they are different, we can generalize the target density function as:

$$\phi_t = (\phi_{t_2} - \phi_{t_1})_+ + \alpha(\phi_{t_1} - \phi_{t_2})_+, \tag{7.9}$$

where α is a workload ratio of disassembly to assembly and $(\cdot)_+$ represents positive only. From now on, we set $\alpha = 1$. The demanding mass is extended to two types: for assembly

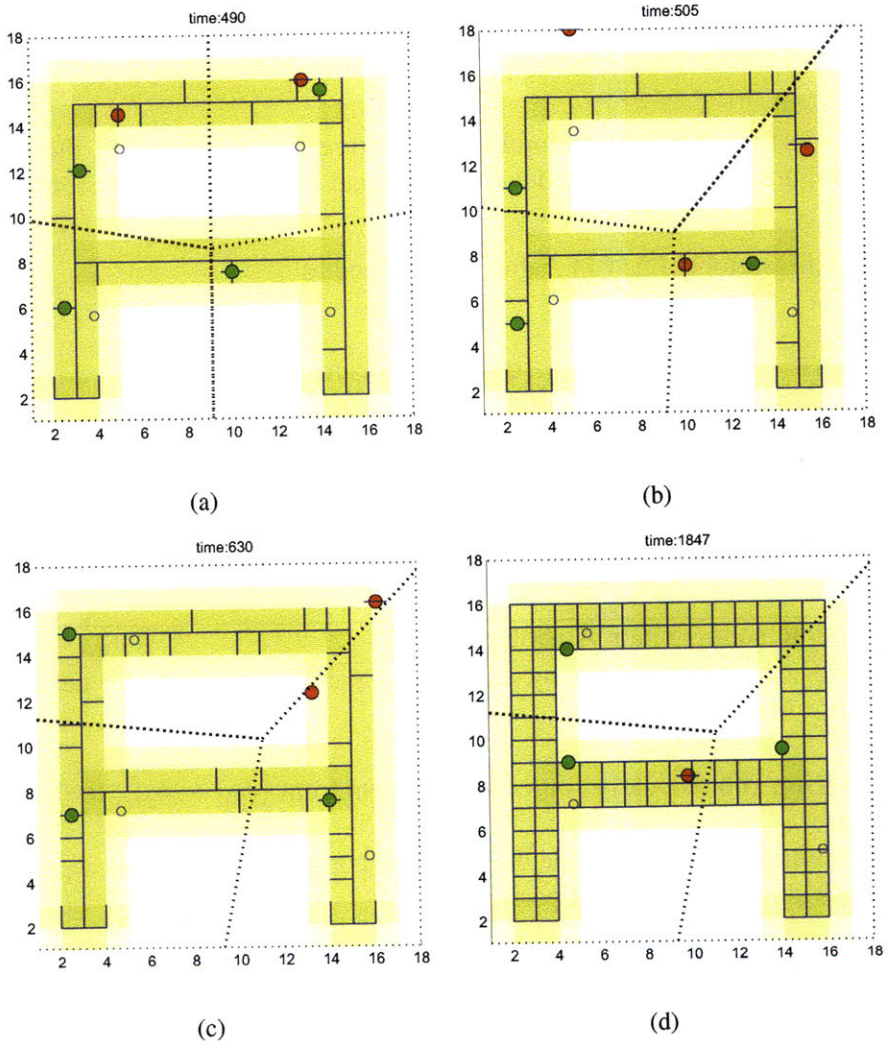


Figure 7-3: 4 assembling robots are constructing the bridge and one of them fails at time 500. Green circles are assembling robots, and red ones are delivering robots. Blue hollow circles are the virtual center of V_i. After failure, the remaining robots reconfigure their Voronoi regions adaptively and finish the construction.

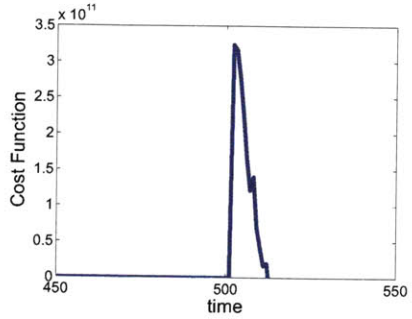


Figure 7-4: Cost function of the simulation in Figure 7-3. At time 500, the cost rises up because of failure, however settles down by the equal-mass partitioning controller.

($\Delta M^a_{V_i}$) and disassembly ($\Delta M^d_{V_i}$) , which are defined as

$$\Delta M^a_{V_i} = \int_{V_i} (\phi_{t_2}(\mathbf{q}) - \phi_{t_1}(\mathbf{q}))_+ d\mathbf{q} - \int_{V_i} \rho_a(\mathbf{q}) d\mathbf{q}, \tag{7.10}$$

$$\Delta M^d_{V_i} = \int_{V_i} (\phi_{t_1}(\mathbf{q}) - \phi_{t_2}(\mathbf{q}))_+ d\mathbf{q} - \int_{V_i} \rho_d(\mathbf{q}) d\mathbf{q}, \tag{7.11}$$

where ρ_a is the density function of the built structure and ρ_d is of the disassembled structure.

Algorithm 10 Reconfiguration Algorithm

1: Place the assembling robots by equal-mass partitioning with the density function $|\phi_{t_2} - \phi_{t_1}|$ in Q
2: **repeat**
3: **delivering robots:** carry source components from $(\phi_{t_1} - \phi_{t_2})_+$ to the assembling robots
4: **assembling robots:** assemble the delivered components in $(\phi_{t_2} - \phi_{t_1})_+$
5: **until** task completed *or* out of parts

Assembly Algorithm

The state machine used for the assembling robot in [65] is adjusted for reconfiguration. The robot has the following states:

- IDLE
- WAITING: waiting for a new component or request for a part
- MOVING_ASSEMBLY: moving to the optimal location to add the part

- ASSEMBLING: adding the component to the assembly
- MOVING_DISASSEMBLY: moving to the optimal location to detach the part
- DISASSEMBLING: removing the component and hand over it to a delivering robot

The last two states are added to the state machine in [65] for disassembly.

Algorithm 11 shows the details of the state machine for disassembly. The state machine for assembly is in [65]. When reconfiguration starts, an assembling robot initializes the parameters R, E, ρ_a, ρ_d and changes its state to WAITING. Recall that each robot has a local graph representation $G = (R, E)$ of the already built local substructure by itself and neighbors. If it receives a request for disassembly from a delivery robot, it finds the optimal location to remove a truss element in $(\phi_{t_1} - \phi_{t_2})_+$. The optimal location is chosen as an edge with the maximum demanding mass for disassembly. The robot moves to the location by setting the state to MOVING_DISASSEMBLY. In the MOVING_DISASSEMBLY state, an assembling robot moves to the target location $\mathbf{t}$ and changes the state to DISASSEMBLING when it arrives. Then it detaches the truss element and hand it over to the delivery robot. After disassembly, it updates the parameters such as R, E, ρ_d. The state goes back to WAITING.

7.3.1 Delivery Algorithm

Delivering robots also operate by an adjusted state machine from [65]. Each robot has the following states:

- IDLE
- ToSOURCE: moving to a picked point in $(\phi_{t_1} - \phi_{t_2})_+$
- ToTARGET: moving to a picked point in $(\phi_{t_2} - \phi_{t_1})_+$
- ToASSEMBLY: delivering the element to an assembling robot
- ToPICKUP: moving to get a new element from an assembling robot
- PICKING: getting the element from the assembling robot

Algorithm 11 Control Algorithm of assembling robots

STATE: IDLE

$R \leftarrow nodes \in \phi_{t_1}(V_i)$, $E \leftarrow edges \in \phi_{t_1}(V_i)$

$\rho_a(\mathbf{q}) = 0, \rho_d(\mathbf{q}) = 0$

state=WAITING

STATE: WAITING

if receive a request for disassembly **then**

  e =findOptimalEdge$(R, E, (\phi_{t_1} - \phi_{t_2})_+, \rho_d)$

  $\mathbf{t} = \mathbf{q}_{(node_1(e)+node_2(e))/2}$

  state=MOVING_DISASSEMBLY

end if

STATE: MOVING_DISASSEMBLY

if reached $\mathbf{t}$ **then**

  state=DISASSEMBLING

else

  move to $\mathbf{t}$

end if

STATE: DISASSEMBLING

disassemble the material

update $\rho_d(e)$

$E \leftarrow E - e$

$R \leftarrow R - \{node_1(e), node_2(e)\}$

hand over the material to the delivery robot

state=WAITING

The last two states are for disassembly.

Algorithm 12 describes the details of the state machine. Instead of obtaining a source component from a source cache as in [65], a delivering robot gets it from the redundant structure $(\phi_{t_1} - \phi_{t_2})_+$ and carries it to the unfilled structure $(\phi_{t_2} - \phi_{t_1})_+$. Given an initially empty state, a delivering robot changes its state to ToSOURCE and picks a possible source location with respect to the probability density function $(\phi_{t_1} - \phi_{t_2})_+$. This probabilistic choice has already been used for finding an assembly location in [65], and we use the same method to pick a source component here. The state ToSOURCE ends when the robot reaches the chosen location and switches to ToPICKUP. In the state ToPICKUP, the robot figures out an assembly robot with the maximum demanding mass that is a sum of $\Delta M^a_{V_k} + \Delta M^d_{V_k}$. To ensure there is a source component to be disassembled, the assembly robot should have positive demanding mass for disassembly ($\Delta M^d_{V_k}$.) If the assembling robot has the state WAITING, then it requests disassembly and moves to the robot, switching the state to PICKING. The delivery robots waits for the assembling robot to finish disassembly and receives the new truss element, changing the state to ToTARGET. The assembly procedure for the state ToTARGET and ToASSEMBLY has been explained in [65].

7.3.2 Implementation

Figure 7-5 shows snapshots of reconfiguration from an A-shaped bridge (Figure 7-5(a)) to an M-shape (Figure 7-5(b)). 4 assembling robots and 4 delivery robots are deployed. We can see the density function $|\phi_{t_2} - \phi_{t_1}|$ for equal-mass partitioning has a cross-like shape (the yellow region without the truss and the truss outside the yellow region in Figure 7-5(b).) The partitioning results in new Voronoi regions as in Figure 7-5(c), and the delivering robots carry a truss element from redundant truss to the yellow region that is not filled by the truss yet.

7.4 Multiple types of source components

Figure 7-6 shows snapshots of the simulation of building the A-shaped bridge with two types of truss elements: *side* and *diagonal*. The density function is a simple sum of that for

Algorithm 12 Control Algorithm of delivering robots

STATE: IDLE

state = ToSOURCE

$\mathbf{t} \sim (\phi_{t_1} - \phi_{t_2})_+$

STATE: ToSOURCE

if reached $\mathbf{t}$ **then**

    state = ToPICKUP

else

    move to $\mathbf{t}$

end if

STATE: ToPICKUP

communicate with robot r_i s.t. $\mathbf{q} \in V_i$

$deliveryID = \text{argmax}_{(k=i,j\in\mathcal{N}_i),\Delta M^d_{V_k}>0} \Delta M^a_{V_k} + \Delta M^d_{V_k}$

if r_i = WAITING **then**

    send a disassembly request to r_i

    state = PICKING

    $\mathbf{t} = \mathbf{p}_{deliveryID}$

end if

STATE: PICKING

if reached $\mathbf{t}$ and get a truss element **then**

    state = ToTARGET

    $\mathbf{t} \sim (\phi_{t_2} - \phi_{t_1})_+$

else

    move to $\mathbf{t}$

end if

STATE: ToTARGET

if reached $\mathbf{t}$ **then**

    state=ToASSEMBLY

else

    move to $\mathbf{t}$

end if

STATE: ToASSEMBLY

communicate with robot r_i s.t. $\mathbf{q} \in V_i$

$deliveryID = \text{argmax}_{(k=i,j\in\mathcal{N}_i),\Delta M^a_{V_k}>0} \Delta M^a_{V_k} + \Delta M^d_{V_k}$

$\mathbf{t} = p_{deliveryID}$

if reached $\mathbf{t}$ & *state* of r_i = WAITING **then**

    pass the material

    state = ToSOURCE

else

    move to $\mathbf{t}$

end if

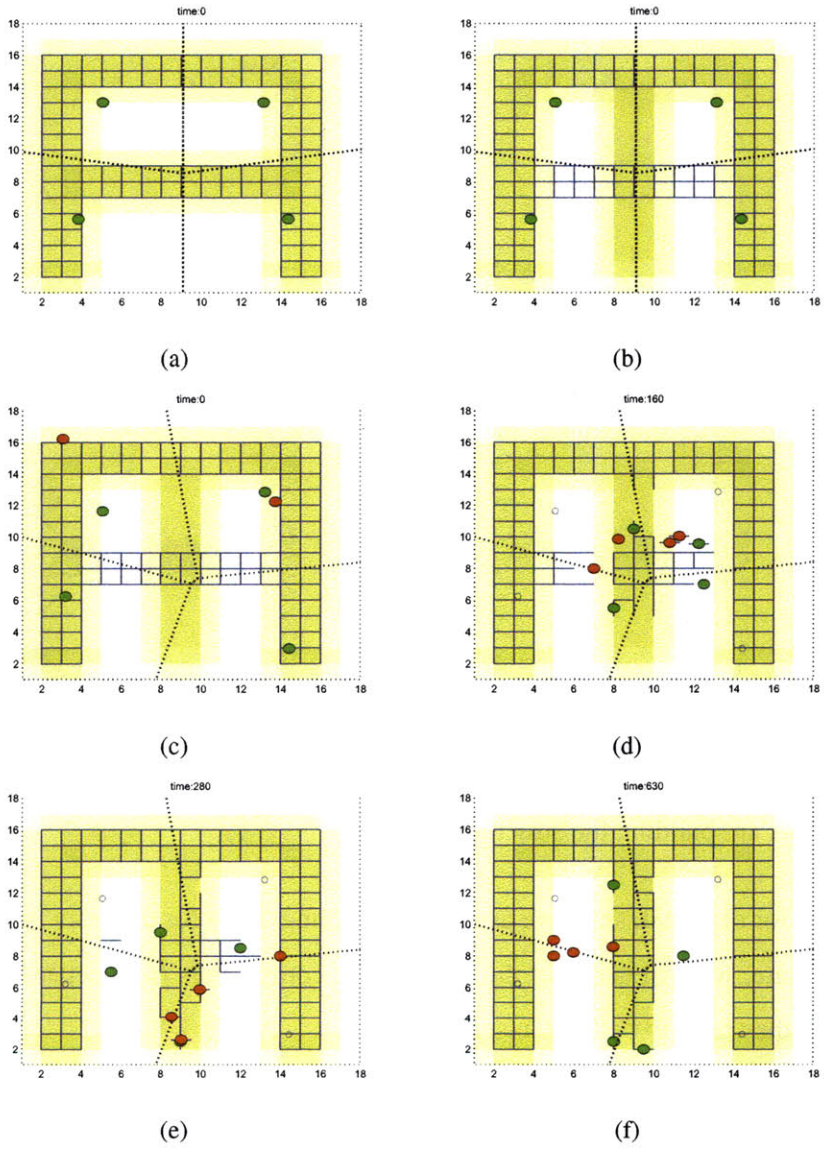


Figure 7-5: Reconfiguration from the A-shaped bridge to the M-shaped bridge. 4 assembling robots and 4 delivering robots are used. (a) Completion of building the A-shaped bridge (b) New density function for the M-shaped bridge (c) Equal-mass partitioning for difference between the density functions (d-f) Reconfiguration

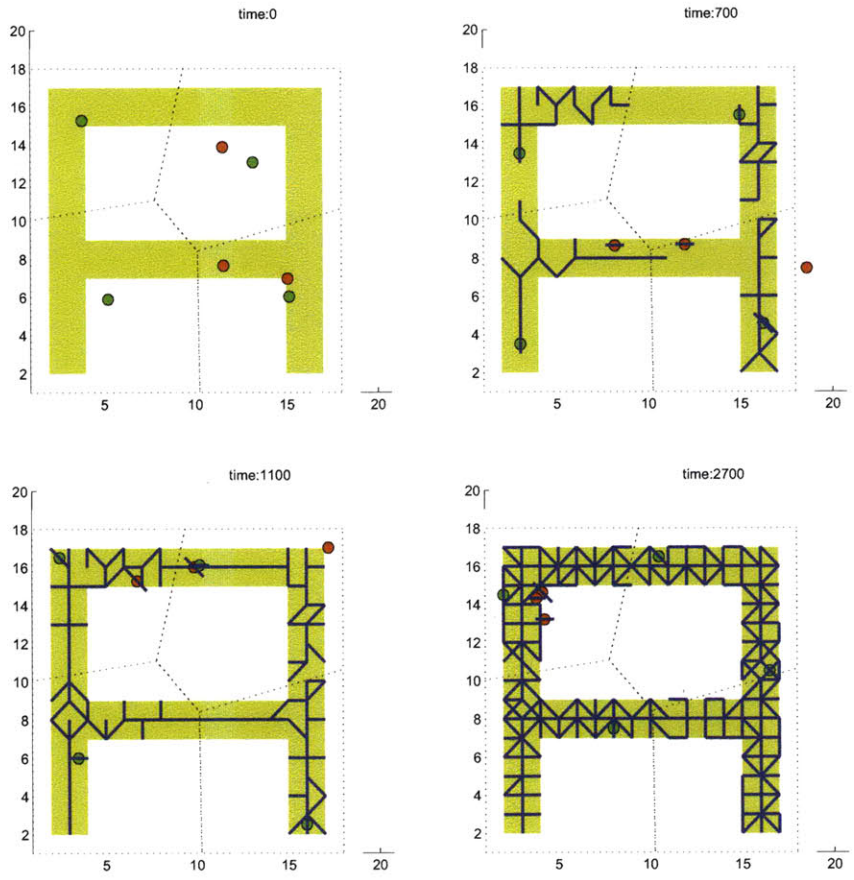


Figure 7-6: A-shaped bridge with two types of truss elements. 4 assembling robots and 4 delivery robots are building the structure.

side and that for diagonal, since we assume assembling times for them are the same.

In the future, we will consider the case source components have dependency on each other so that they have to be built in some order.

7.5 Adaptation to human input

Next, we wish to enable our system to take input from and adapt to human input during operation. For example, when a failure such as misalignment between parts happens because of a lack of fine manipulation, the robot can call a human helper to fix the problem rather than try to solve it by itself.

To implement human input, a human robot interface is essential for a human to com-

municate with robots. Fortunately, a mesh network is required for our system and can be implemented as shown in Chapter 8, and a human helper can benefit from the existing network by bringing a device which can be connected to the network and send/receive packets.

7.5.1 Human input

Humans may take either the role of a delivery robot or the role of an assembly robot. Human delivery is straightforward. One can pretend to be a delivery robot as long as one can send the same message as a delivery robot is supposed to send to an assembly robot. An assembly robot is not able to tell the difference between human delivery and robot delivery if the communication device of a human worker is compatible with the controller (UDP communication by a netbook in our hardware implementation) of a delivery robot. In our hardware implementation, this can be done by informing an assembly robot of the part that has been delivered by a human and receiving acknowledgement from the assembly robot. Before the delivery, a human must confirm the assembly robot has positive demanding mass of the part, and this can also be done by listening to the demanding mass as a delivery robot does in our implementation. In this way, a human intervention speeds up the process when there is a bottleneck of supply.

Human assembly can not be done as transparently as human delivery. Since every assembly should be checked by an assembly robot which has the assembly location in its Voronoi partition, human assembly should be reported to the assembly robot via an additional protocol. Implementation of human assembly is also straightforward. Once a human assembled a part, one sends a message including the location of assembly, and the corresponding assembly robot updates its map (ρ and Φ^c) as in ASSEMBLING state. To avoid confliction, a human must check if the assembly robot has WAITING state.

7.5.2 Guide for build order

Assembly robots choose the order of construction by Algorithm 7. Human input can be used to change the order when priority arises during construction. We define a set of prior-

itized node R_p as locations at which parts should be assembled prior than other locations. The prioritized nodes are selected by human input during construction by sending out a message in a mesh network. Algorithm 7 has been modified to Algorithm 13 so that it first chooses an edge which is connected to the prioritized nodes. If there is no edge that is adjacent to the prioritized nodes, then robots continue the original Algorithm 7.

Algorithm 13 Finding the Optimal Edge to Build with the Prioritized Nodes

1: **if** $R_p \neq \varnothing$ **then**
2: $r \leftarrow R_p$
3: $e \leftarrow edges(r) \cap (\phi_t(edges(r)) - \rho(edges(r)) > 0)$
4: **if** no more possible edges to r **then**
5: $R_p = R_p - r$
6: **end if**
7: **return** e
8: **else**
9: **return** e=findOptimalEdge(R, E, ϕ_t, ρ) (Alg. 7)
10: **end if**

Figure 7-7 shows implementation of Algorithm 13. In the beginning, the two lower robots have the prioritized nodes on the base, therefore they start construction from the base. After a while, new prioritized nodes are given at the borders of the Voronoi regions, and all the robots focus on assembling the trusses at the prioritized nodes before they finish the structure. In the simulation, the prioritized nodes are given by mouse clicks in MATLAB GUI.

7.5.3 Reconfiguration during construction

The reconfiguration algorithms in Algorithm 11 and 12 can be used to change a blue print during construction. We assume we have the ability to communicate to all assembly robots the change of the density function. The assembly robots respond to the human input and reallocate their work partitions. The delivery robots have two options for the next parts to be delivered: the source cache and the components from the old structure that are no longer necessary because of reconfiguration of the density function. Algorithm 14 shows the modified control algorithm for delivery robots. Note that only the controller for the IDLE state of the state machine has been changed. Unlike the reconfiguration in the previous

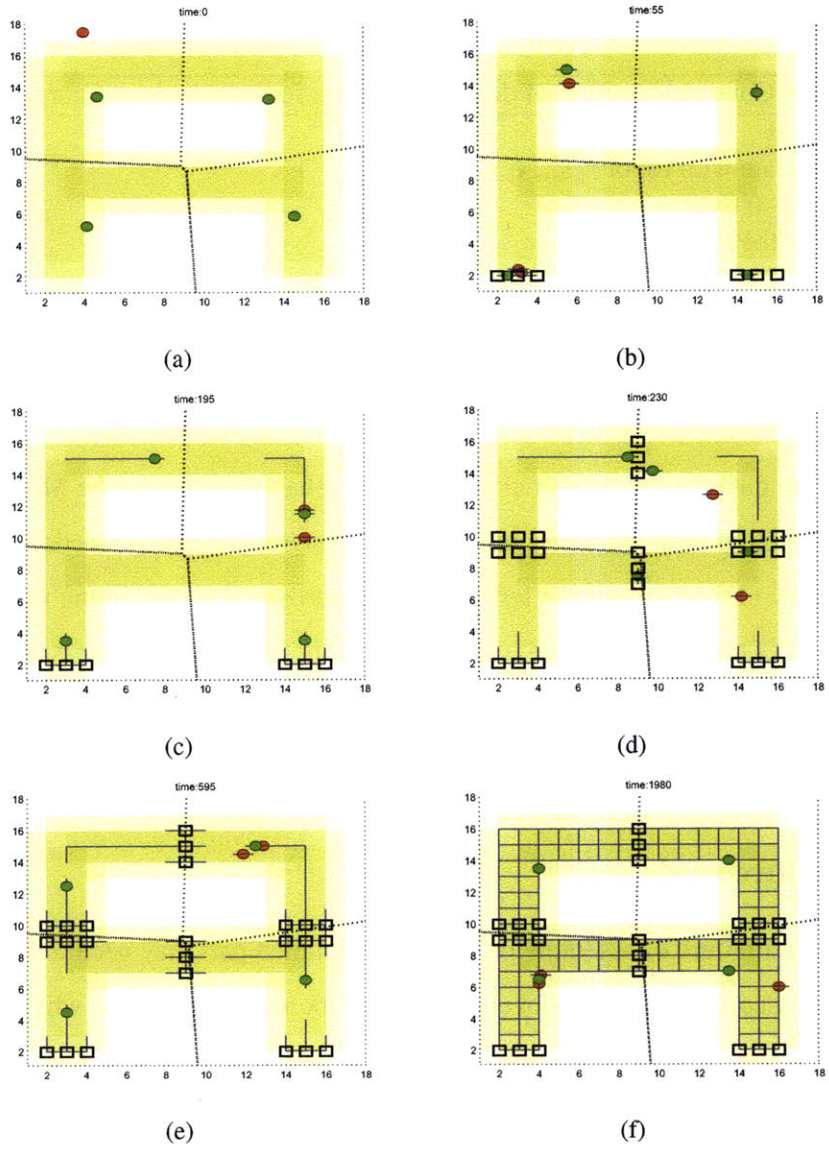


Figure 7-7: Snapshots of building the A-shaped bridge by 4 assembly robots. The black squares are the prioritized nodes which are selected by human input. Between (a) and (b), the two lower robots are given the six priority nodes on the base, and the nodes are built first as shown in (c). More prioritized nodes are given at the borders of the Voronoi regions between (c) and (d). Each of six nodes are prioritized at each side, and each of three nodes are at the top and the middle. The nodes are also built prior to other nodes as shown in (e) and the robots continue the rest of the structure in (f).

section, the robots do not know how many redundant parts there are since the goal structure has not been done yet. Therefore, a delivery robot obtains a redundant part by changing the state to ToPICKUP whenever a neighboring assembly robot has any redundant parts.

Algorithm 14 Control algorithm of delivering robots for human reconfiguration

STATE: IDLE

1: communicate with robot r_i s.t. $\mathbf{q} \in V_i$
2: **if** $\Delta M^d_{V_i} > 0$ **then**
3: *state* = ToPICKUP
4: **else**
5: *state* = ToSOURCE
6: $\mathbf{t} = \mathcal{S}$
7: **end if**

Simulation data for the case of changing the A-shaped bridge to the M-shaped bridge during construction is shown in Figure 7-8. We change the density function twice during the simulation. (1) We cut off the left half of the horizontal bridge in the middle and add the upper half of the vertical bridge. (2) The rest of the horizontal bridge is deleted and the rest of the vertical bridge is added. After the changes, equal-mass partitioning based on the updated density function is followed to equalized the changed workload.

7.5.4 Failure mode

When there is a failure of assembly or delivery, the failed part may need to be removed or correctly re-assembled. If the recovery is beyond the robot capability, which is plausible since we target an inexpensive system with the minimal functionality, a robot can call for a human help through the network. Emergency protocol should be implemented so that the message can travel to a human in a global way. Our future work includes the hardware implementation of this protocol.

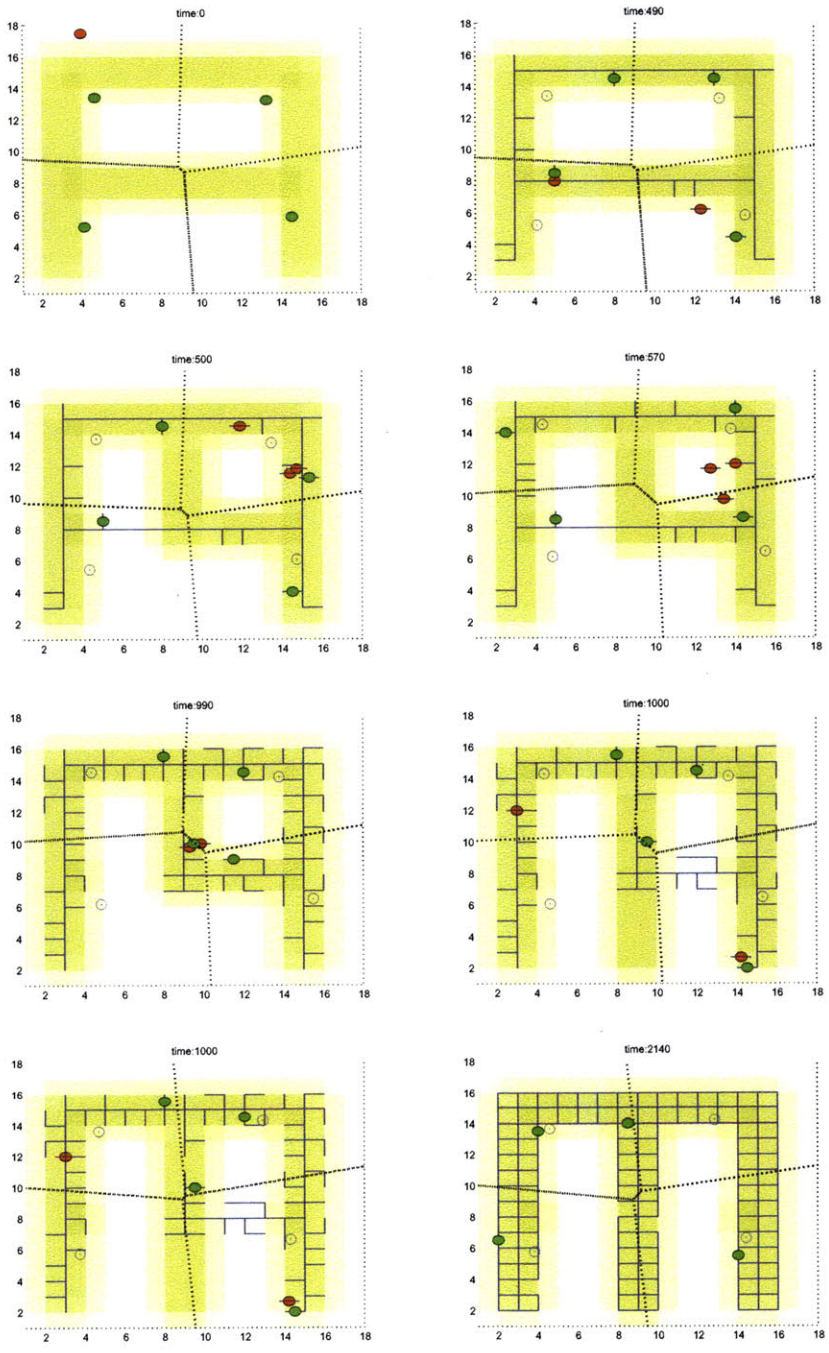


Figure 7-8: The A-shaped bridge is transformed to the M-shaped bridge by two human inputs each of which is followed by equal-mass partitioning. At each change, the redundant parts are used another source cache. The blue hollow circles are the virtual centers.

Chapter 8

Theory to Practice: Experiments

In this thesis, we focus on delivery and assembly experiments. Experimenting equal-mass partitioning is left for future work. Similar algorithms have been implemented before in our prior work [99].

8.1 Experimental System

Our hardware system consists of a team of mobile manipulators, 3D-printed smart parts each with an embedded communication device, and a VICON motion capture system. The robots operate on a square area, and a source cache of trusses and connectors is located at the side of the workspace (See Figure 8-16). The trusses and connectors are manually supplied to the cache during experiments. In order to help grasping, each smart part contains an IR beacon and a battery designed to communicate with the robots. The robots localize using the motion capture system which broadcasts 3D poses over a mesh network.

8.1.1 Mobile manipulator

The robot consists of a commercially available iCreate mobile platform and a CrustCrawler robotic arm with a custom chassis as shown in Figure 8-1. Specifications of each component are in Table 8.1. An instrumented gripper which contains an IR communication transceiver is attached to the arm. The gripper is contoured so that its closing aligns a

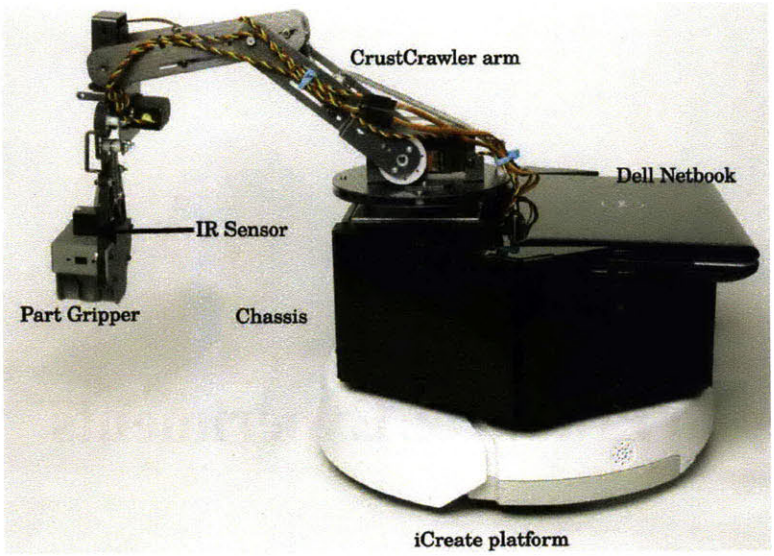


Figure 8-1: Side view of robot hardware with the Crustcrawler arm. From a fixed base, the arm allows for grasping an object on the ground in a half-arc in front of it with a depth of about 20cm.

Mobile		iRobot iCreate
Arm	Model	CrustCrawler SG5-UT
	DoF	4
	Reach	0.5 m
	Payload	0.6 kg
Communication		IR, UDP, xBee

Table 8.1: Specifications of the robot

grasped part. The special design helps the gripper with reliably grasping parts despite of centimeter-scale uncertainty in a position of the parts, by passively aligning the gasping point of the parts into a right orientation as the gripper closes. The robot has three communication protocols: IR, UDP and xBee, which are used for communication with the smart parts, other robots and motion capture system, respectively. The controller is a Dell Inspiron Mini 10s netbook which runs a Java-based controller. The netbook has a WLAN card for inter-robot communication over a wireless network, and the battery of the netbook also powers the IR sensors and the xBee module.

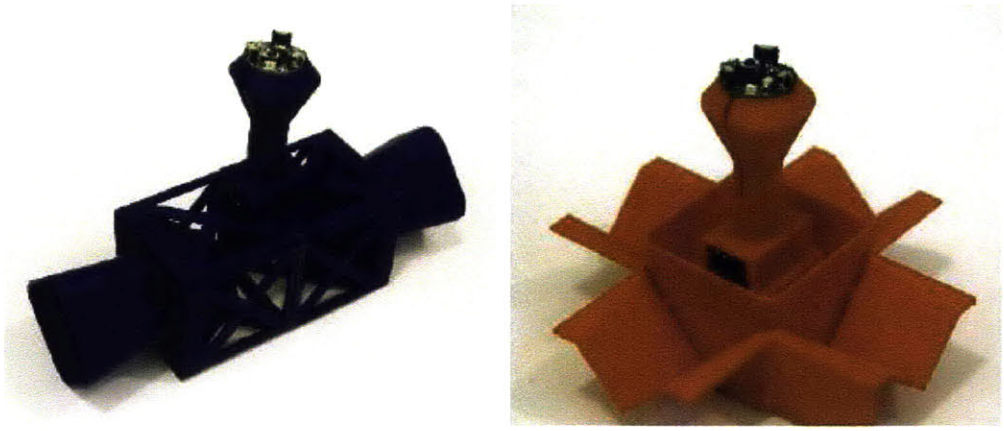


Figure 8-2: Smarts parts to be delivered: (LEFT) a blue truss (RIGHT) a red connector

8.1.2 Smart parts: Instrumented trusses and connectors

Smart parts enable grasping for robotic delivery and assembly via IR communication. We explore the use of communication as an alternative to using computer vision for part identification and grasping. The IR communication devices are instrumented as shown in Figure 8-3 on the robots and within each parts. This allows a robot and a part to interact with each other. A part can guide a robot to its location and tell the robot its part type.

Figure 8-2 shows two types of the smart parts: truss and connector. The angled design of the connecting points aims to compensate for position uncertainty between the smart parts during assembly. The parts interlock each other both horizontally and vertically for scaffold-like structures. The connector is capable of linking 6 trusses in the North, South, East, West, Up, and Down directions. Figure 8-4 shows a cube built from 8 connectors and 12 trusses. Only centimeter scale accuracy is required for assembly, relying on the contoured design of the mating surfaces to fall into place precisely. Every part has the specially designed grasping point that can be passively aligned to a fully constrained position despite up to 2cm of misalignment. With a rechargeable 3.7v 210mAh lithium polymer battery, the parts weigh 60 grams. The truss is 18 cm long.

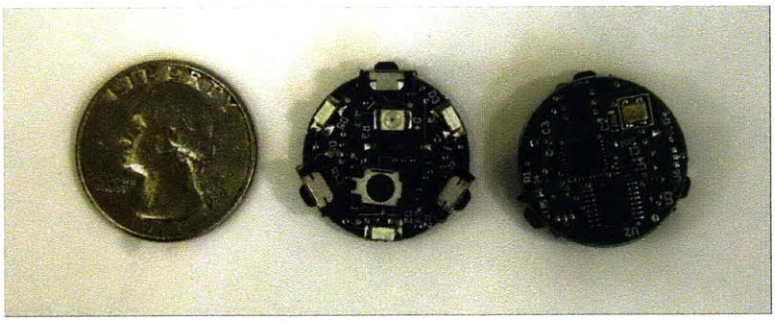


Figure 8-3: The small IR communication modules on a PCB that can be embedded in parts to create a smart environment for the robots to sense. Figure reproduced with permission [33]

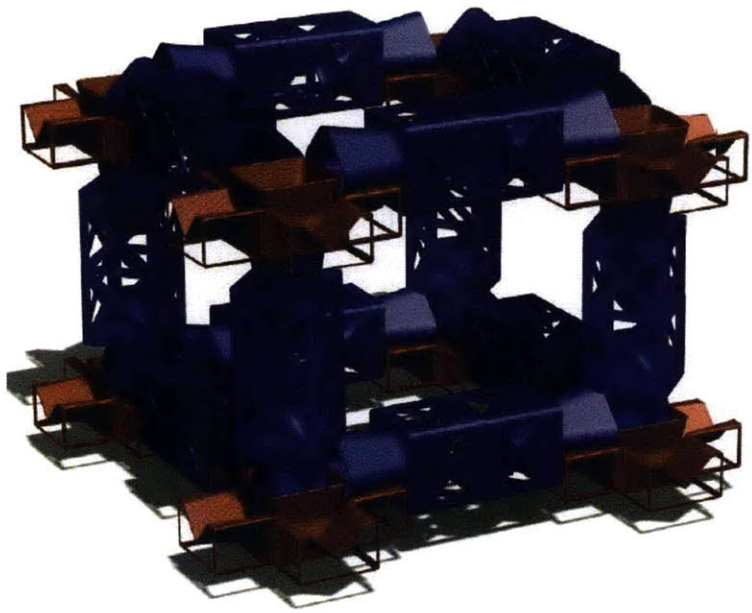


Figure 8-4: This 3D-rendered image of a cube is constructed from 8 junctions, and 12 struts. Picture reproduced with permission [33].

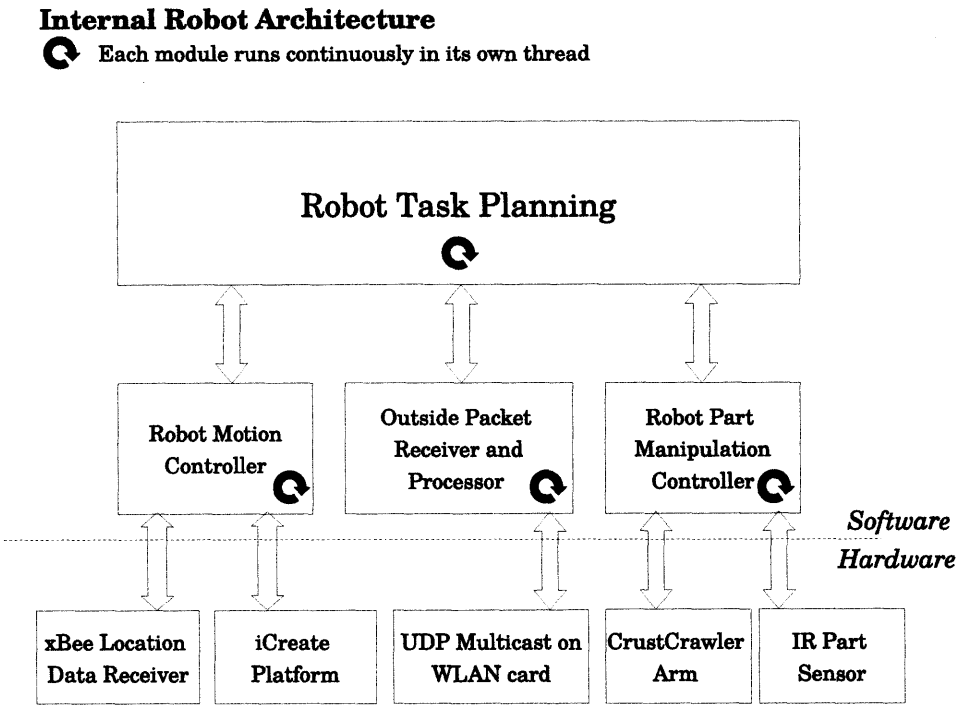


Figure 8-5: The hierarchical software architecture of the robot platform. Reprinted with permission from [17].

8.1.3 Infrastructure for localization and communication

For localization, the robots receive precise poses from a Vicon motion capture system which provides the 2D positions and the rotational headings with accuracy to millimeter and milli-radian at 10 Hz using a xBee radio frequency wireless mesh network. The robots use a UDP multicast channel on the local network for communication. The UDP packet contains a logical time-stamp, a robot ID, a current position, and a current target robot of delivery. The robots also broadcast their states: whether they are currently carrying or dropping off a part, which type of parts they are carrying, where they are carrying this payload to, and the location of any other known placed parts.

8.1.4 Software Architecture

The software architecture of the main controller which runs in the netbook is structured hierarchically and modularized. The highest level planners are derived from the same super

planner. This modularity leads to *assembly* and *delivery* planners, one of which each robot chooses according to its role to control the robot functions as shown in Figure 8-5.

All the software codes are written in Java and each module runs its own thread. The planner thread mainly controls manipulation and navigation. The planner gives the navigation module a destination pose and obstacles, such as other robots and parts on the ground. After navigation, The arm module receives two commands from the planner: *pick up the part* or *put down the part.* The planner decides where and when to move and manipulate parts given the information received by the communication module. The communication module provides the most up to date information for the planner to make a next decision on navigation and manipulation. Under the planner, three modules handle low level control for the mobile, the arm, the manipulating IR sensors, and the communication messaging hardware.

8.2 Extended State Machines

Implementing Algorithm 1 on the robot system requires revisiting its assumptions with respect to what can be measured, implemented, and computed efficiently, and making corresponding changes to control loops. The main differences between the theory and the practice are listed in Table 8.2. The most important components are manipulation and navigation, used both for assembly and delivery.

8.2.1 Navigation

The original algorithm did not consider any collision between robots and already built structures, and we extend the algorithm to allow the robots to physically move around other robots and parts by passing required data in the communications messages. The navigation software module shown in Figure 8-6 receives commands from the planner and moves the robot as close to a desired pose as possible. The A-star algorithm drives a delivery robot to approach a destination location on a grid map which divides the square area into 10 $\times$ 10 cm mesh. A proportional motion controller is appropriate for the iCreate platform. The navigation module checks collision avoidance in real time, and prevents a robot from

Experiment	Controller from [65]
• Nonholonomic robot dynamics arises position errors and turning delays	• Holonomic robot
• Noisy measurement of global position	• Knowledge of exact global position
• Robots with volume and dynamics, path planning required	• Robots are point masses
• Collision avoidance algorithm required	• Robots pass through the environment
• The next part to be delivered is dependent of the current structure	• No dependency between trusses and connectors
• Pickup causes a bottleneck	• Picking up parts from supply cache takes very short time
• IR beacons for communication between robots and materials	• Pin-point knowledge of types and locations of materials
• UDP messaging system using acknowledgements and logical time to recover packet loss	• Synchronous communication for complete information about surroundings
• Asynchronous propagation of information	• Immediate update of information from neighbors
• Hardware failure causes part to be dropped	• Parts never lost or dropped on map

Table 8.2: Controller from [65] Vs. Experiment

moving to a location blocked by an obstacle or other robots.

8.2.2 Manipulation

Manipulation is used for obtaining a part from the source cache, handing off the part to an assembly robot and placing the part for assembly. In each case, the planner uses the arm module to find, pick up and place the part. Algorithm 15 is the search-and-pick motions of a delivery robot based on the robot-part communication via the IR beacon.

The field of view of the IR sensor of the arm can be widened and narrowed by opening and closing the gripper, and the arm finds parts by iteratively scanning smaller and smaller areas for an IR signal. First, the arm points the sensor directly at the ground to avoid detecting any parts that are out of its reach. The first pass of an area requires scanning by moving the entire arm in a 180 degree half circle around the front of the robot. The arm module closes the gripper and narrows the sensors cone of view until it no longer sees the part, and re-scans the small area until it finds the part again. This process continues until the arm has been moved close enough to the part to grip the top of it and pick it up.

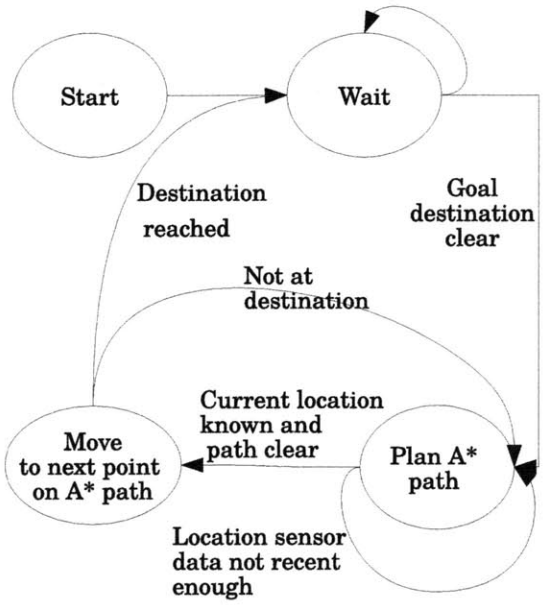


Figure 8-6: The motion planning FSM of the robot software. Reprinted with permission from [17].

Algorithm 15 Arm Manipulation Part Search Algorithm

```
1:  repeat
2:     Open gripper for wide FOV
3:     while IR sensor does not see part do
4:        Arc scan back and forth π radians
5:     end while
6:     startTheta = current arm position
7:     while IR sensor still sees part do
8:        Radial scan forward.
9:     end while
10:    endTheta = current arm position
11:    Narrow gripper field of view
12:    while IR sensor does not see part do
13:       Move arm in and out along radius while arc-scanning
14:       between startTheta and endTheta radians
15:    end while
16:    Open gripper wide.
17:    Lower arm on top of part
18:    Close gripper
19: until Arm closed over part
```

Narrowing down the possible locations of the part allows the arm to fine tune its signal to be within 2cm of the top of the part. The arm confirms pickup by receiving a response from the parts IR chip while the arms gripper, hold the part, is pointed up in the air. Snapshots of grasping is shown in Figure 8-13. This search-and-pick algorithm is also used for hand-off of an assembly robot with smaller search angles ($\pm 10 degree$).

8.2.3 Communication

The communication module runs constantly in its own thread to provide the most recent data to the task planner. The module maintains the latest state of all the robot broadcasting in the signal range and stores the most recent message logical time of which is determined by packet time-stamps implemented using distributed logical time. The module also broadcasts out its own state on the same channel and keeps track of parts which other robots have reported placing down on the field of construction already so that the robot avoids them during navigation. Finally, for a handshake between two robots, the communication module keeps track of parts expected by an assembly robot, whether a delivery robot has delivered them yet, and whether the target assembly robot has acknowledged the delivery. Since the robot is not equipped with a sensor to see the environment, the record of part movements acts as the sensor. Any robot en route listens to neighboring robots, and it records where parts are being placed and which robots are the targets in order to update progress of construction. Since delivery robots deliver parts based on what other robots in the field of construction demand, the dissemination of knowledge about where parts have been delivered is crucial. Robots need an accurate internal map of the already built site in order to calculate their own demanding mass value or to decide if they move, and the communication module gives this information to the planner.

8.2.4 Delivery

The delivery algorithm is as a finite state machine as shown in Figure 8-7, which follows the theory and accounts for the practical challenges of a multiple robot system including collision avoidance, asynchronous communication, and dependencies between the parts. In

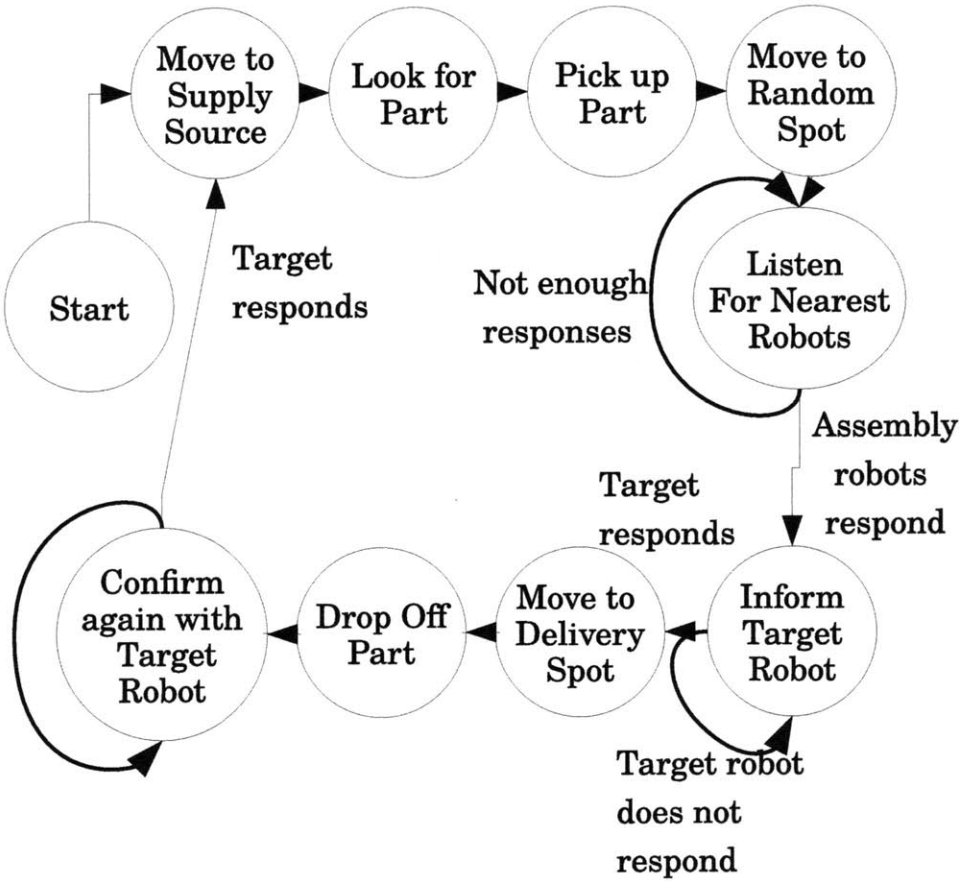


Figure 8-7: The task planning event loop for the delivery robots. The main loop pauses and loops back on itself at points where continuing requires asynchronous communication from other robots. Reprinted with permission from [17].

Algorithm 16 Delivery Robot Part Delivery Algorithm

```
1: repeat
2:    Move to supply source
3:    Pick up part
4:    Move to random location on map
5:    repeat
6:       Listen for demanding mass from nearby assembly robots
7:    until Sufficient network time passes.
8:    Target assembly robot with highest demanding mass.
9:    repeat
10:      Inform target robot of our intent to deliver a part
11:   until We receive a response from target
12:   Move to delivery location
13:   Put down part
14:   repeat
15:      Inform target that part has been delivered
16:   until We receive a response from target
17: until No more assembly robots asking for parts.
```

the theory, the robots have access to perfect information about the all the parameters such as locations and demanding masses of the surrounding robots, and we extend the theory by a fault tolerant and asynchronous communication protocol which allows robots to learn about the surrounding parts and robots. Another assumption of the original algorithm is that the delivery order of parts will not affect the assembly of the structure. Given the hardware of the trusses and the connectors, we use the practical delivery algorithm which incorporates the order of construction in which parts are delivered can be factored into demanding mass calculated. These extensions enable the algorithm to be carried out in the physical system.

The delivery algorithm is enhanced as shown in Algorithm 16 by the sub-modules handling errors. The navigation module checks possible collisions when robots move to the source and to other robots. A delivery robot awaits the source cache to be clear of other robots and it acquires a specialized part from the supply as described in Algorithm 15. Asynchronous communication is used for the robot to find an assembly robot with the maximum demanding mass. After picking a part, the delivery robot listens for seconds to any local robots within the broadcast range. Assembly robots with positive demanding mass broadcast their needs every seconds. At each step of communication, the planner makes acknowledgement before moving on.

A delivery location is chosen so that the distance between the delivery robot and the assembly robot is twice the default arm offset which is measured when a robot places down a part on the ground. Therefore, if the delivery robot can locate itself exactly at the chosen delivery location, the assembly robot would not need to translate but to rotate to pick up the delivered part. If the delivery location is not reachable, the robot figures out the nearest location from the chosen location. After reaching the location, the delivery robot places down and send the assembly robot a packet which includes the part information as well as the delivery location.

8.2.5 Assembly

The assembly algorithm, demonstrated as a finite state machine in Figure 8-8, adds to the original algorithm similar systems as in the the delivery algorithm, including collision avoidance and awareness of the local structure. We also completely replace the computation of the optimal edge to place next, and change the delivery mechanism from a direct handoff to a passing of parts within the general vicinity of the assembly robot. In the original algorithm we compute the least connected edge in our structure and add a part, and also as the model does not consider collision it assumes there is always space for multiple robots to perform a handoff. In our implementation we take advantage of a blueprint, and only allow the placement of parts that both depend on no other parts to hold them up and that do not prevent a robot from reaching the location of an unplaced part. Among these parts, the optimal part is the one that most increases the number of placeable parts in the partition. We also determine handoff points rather that requiring the delivery robot to directly access the assembly robot inside the structure.

The update to the computation of demanding mass mentioned above is directly linked to the update of the computation of the optimal placement of a part. A structure is now represented as a blueprint of interdependent parts, where each part maps to a node on both a directed graph representing the physical dependencies of parts (with an edge from any part to any part that directly requires it to be placed) and an undirected graph of the part's proximity to other parts (with an edge between any two parts within a robot's radius of

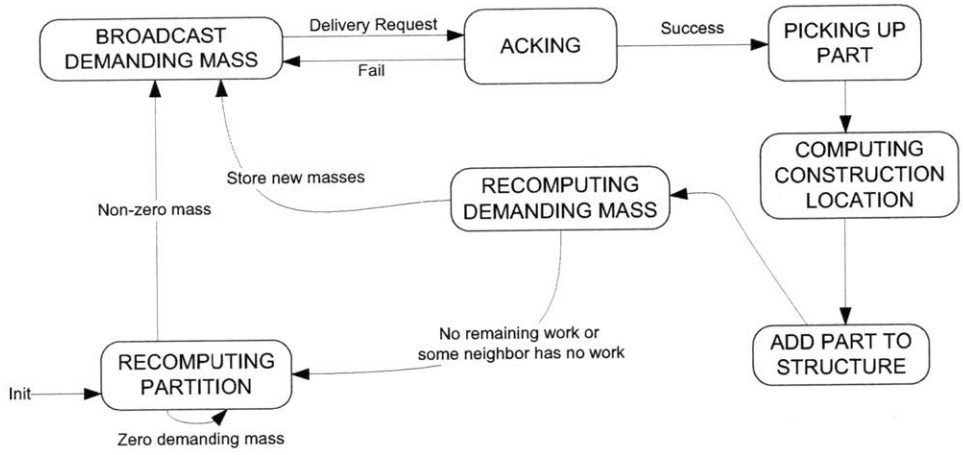


Figure 8-8: The task planning event loop for the assembly robots.

each other). We define a part p as active if it has no parents on the directed graph and that a path exists from every part the robot is responsible for to the edge of the map which does not pass through p. By assuming that the density of parts is bounded, we can provably recompute the set of parts which is active in sublinear time using discrete gradients. As the only parts which can be placed without adding imposable constraints to the task are active ones, we only use active parts when computing demanding mass, meaning the total mass of a partition can both increase or decrease significantly after each placement. We uniquely weight the contribution of a part on the blueprint to the demanding mass by the net change it would have on the size of the set of active parts and break ties by assigning more weight to parts which would remove more constraints from inactive parts, breaking further ties by preferring the centroid of the robot's Voronoi partition. The optimal part placement is determined by the active part with the greatest weight, which means robots place parts in such a way as to allow more parts to be placed, if possible.

Due to the noise in the environment and a decoupling between motor control and localization, accurate locomotion over small distances is nearly impossible, requiring a more careful approach to allow for the accurate placement of parts. By adding a virtual wall to the constraints on the arm, we were able to achieve high fidelity placement despite the noise to our navigation system. Rather than relying on the location of the robot for hand-offs, while in close quarters (on the order of 10 cm), we switch to defining our position by the

location of the robot gripper, allowing for significantly higher robustness with hand-offs.

8.3 Experimental Results

In experiments, we use up to 4 hardware robots, 2 assembly robots and 2 delivery robots in a 5 $\times$ 5 meter rectangle. Poses of all the robots are captured by the Vicon system provided with each robot and a GUI that displays and keep a log of all the activities and communication data.

8.3.1 Coverage on a graph

The equal-mass partitioning algorithm on a graph in Chapter 5 is implemented in our system. We use all 4 robots for coverage as if they are all assembly robots.

We test two graphs: square and an A-shaped bridge as shown in Figure 8-9 and 8-11. The environment is a 3 $\times$ 3 square divided into the 0.1m grids. Nodes are connected when they are adjacent either by side or diagonal. The bridge has two empty spaces as shown in the right figures of Figure 8-11, which do not physically appear in the snapshots of the experiment.

Figure 8-10 and 8-12 shows convergence of the masses during the experiments.

8.3.2 Delivery

Here we test the delivery algorithm only to see the probabilistic deployment and the local gradient-following movement.

Test Delivery Scenario

For evaluation, we use a single blueprint for every test in order to demonstrate different features of the physical system while the number and locations of the assembly robots vary for each run. The blueprint is assumed to have two towers in a 5x5 meter rectangle, and the delivery robots try to deliver as many parts as possible to the assembly robots that are assumed to have the same demanding mass to construct each tower. In each run, the

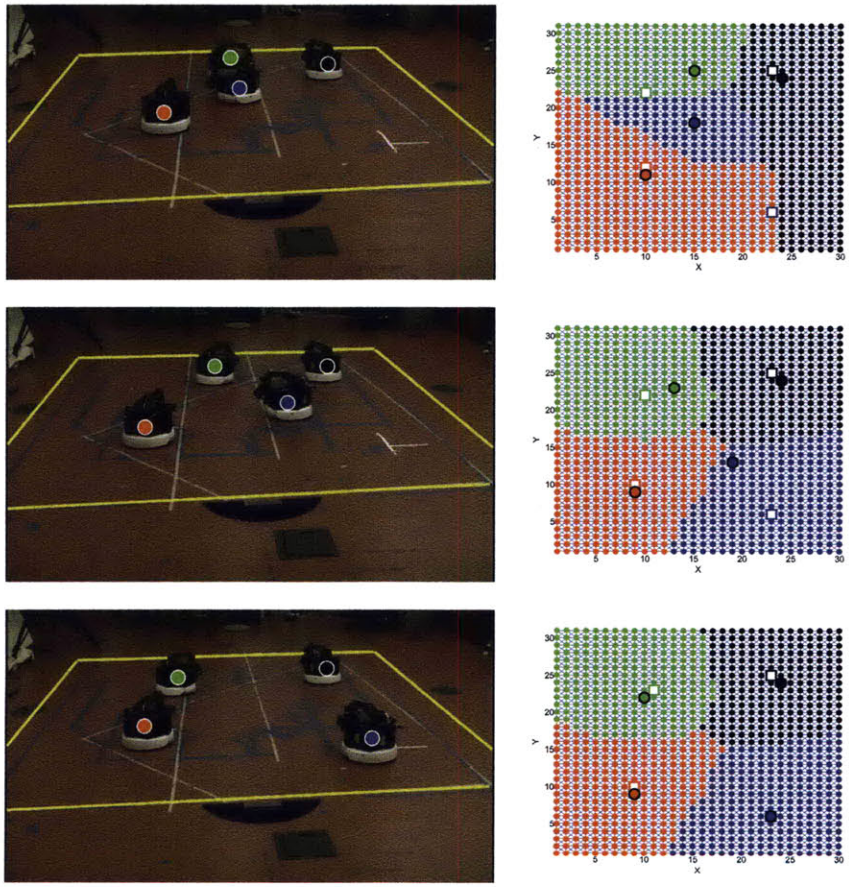


Figure 8-9: Snapshots of equal-mass partitioning on a graph. The right figures show the partitions.

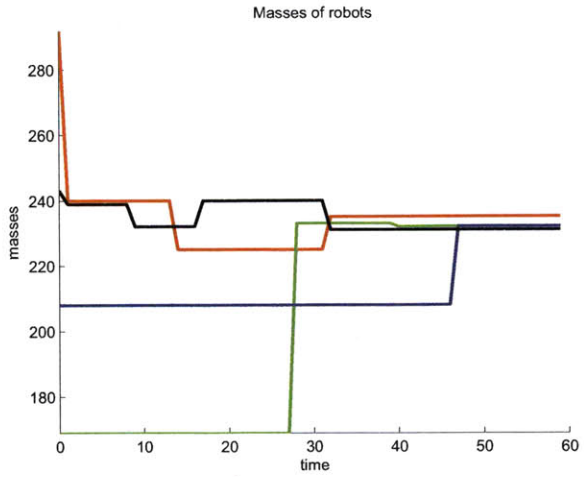


Figure 8-10: Masses of 4 robots during the experiment. All of them converge to a single value.

delivery robots start at random locations. We specialize the delivery robots as one picks up only trusses and the other picks up connectors only. The source cache for parts is located at the origin and manually supplied with the parts. We place both red joint parts and blue truss parts together in the semi-circle shaped supply dock. The delivery robots can sense the difference of parts by communicating with them over IR.

The goal of the scenario is to test load balancing between the assembly robots. Given the blueprint and the assembly robot locations, he delivery robots are supposed to alternate between the two assembly robots. In order to evaluate the system and to see adaptation, we run a basic test and a variation in which an assembly robot quits demanding parts halfway through the test. According to the algorithms, this failure of the assembly robot will lead the delivery robots to adapt and deliver parts only to the remaining assembly robot.

Delivery Experiments

We made twelve runs of the the basic scenario all of which produced the expected alternating behavior of the delivery robots. Both the connector delivery robot and the truss delivery robot switched targets in each delivery and made successful deliveries to the assembly robots, as seen in Figure 8-16. Also as expected, the alternating delivery was responded by

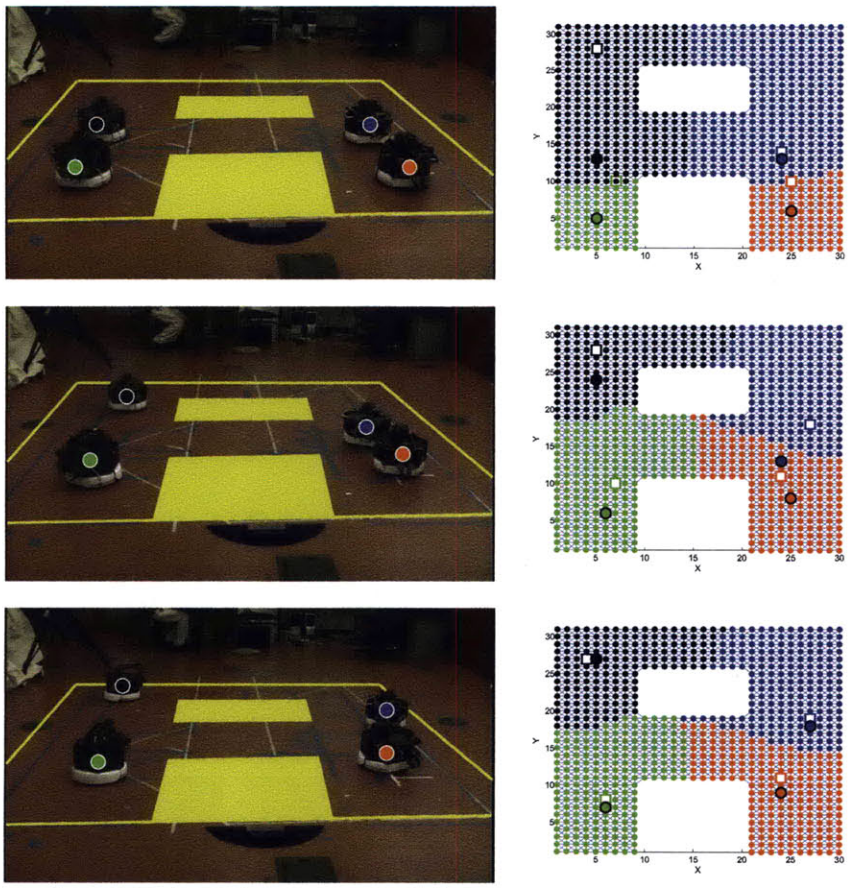


Figure 8-11: Snapshots of equal-mass partitioning on a graph. The right figures show the partitions.

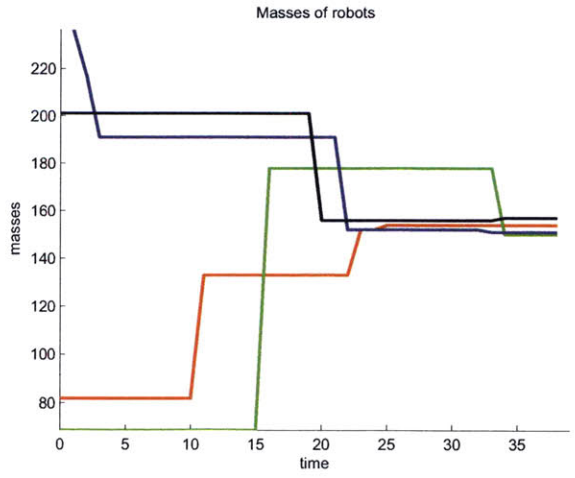


Figure 8-12: Masses of 4 robots during the experiment of partitioning the A-shaped bridge.

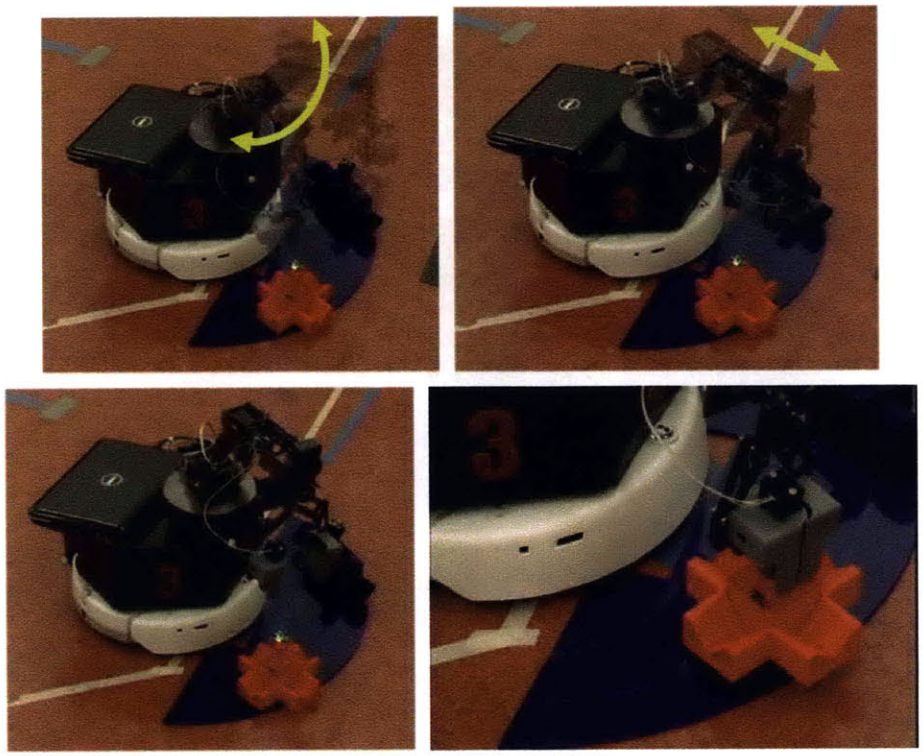


Figure 8-13: Snapshots of grasping. The arm moves along an arc to find a rough position of a part and does fine search by radial motion. Grasping is done after confirming the part.

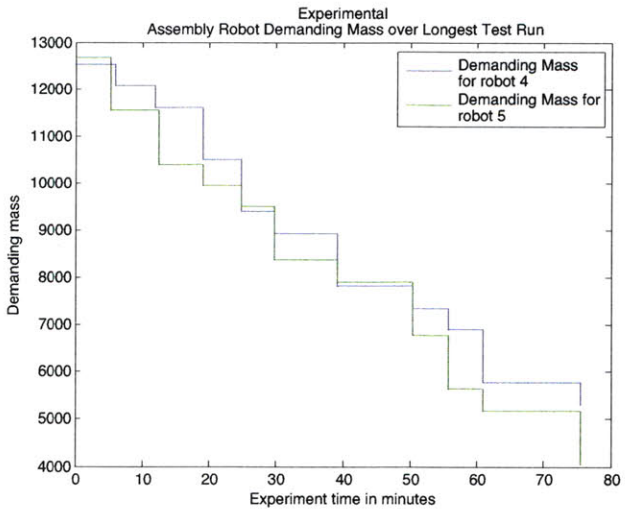


Figure 8-14: The demanding masses of assembly robots, named robots 4 and 5, drops whenever a part delivery occurs. Delivery robots changed targets to whichever robot had the highest demanding mass at the time.

the step-like decay of the demanding masses reported by the assembly robots as shown in Figure 8-14.

For the adaptation test, in order to simulate a failure of an assembly robot, one of the assembly robots was taken off the map during the experiment. The simulated failure promptly resulted in the delivery robots delivering to the remaining assembly robot. In all runs, the communication between delivery and assembly robots confirmed the deliveries and correctly updated the demanding masses of the assembly robots. Over all 12 test scenario runs, the two delivery robots completed 45/48 delivery attempts. Three failed deliveries came from an arm hardware failure on a single robot. A summary of test runs can be seen in Table 8.3.

Run Time Empirical Analysis

Each delivery robot averaged 7 minutes for a round trip delivery, spending much of its time dealing with the supply dock rather than the other robots in the system. The summary is in Table 8.3. The robots spent a significant amount of time parked in the supply dock, searching for parts: the robotic arm requires an average of 2.75 minutes (32% total time)to

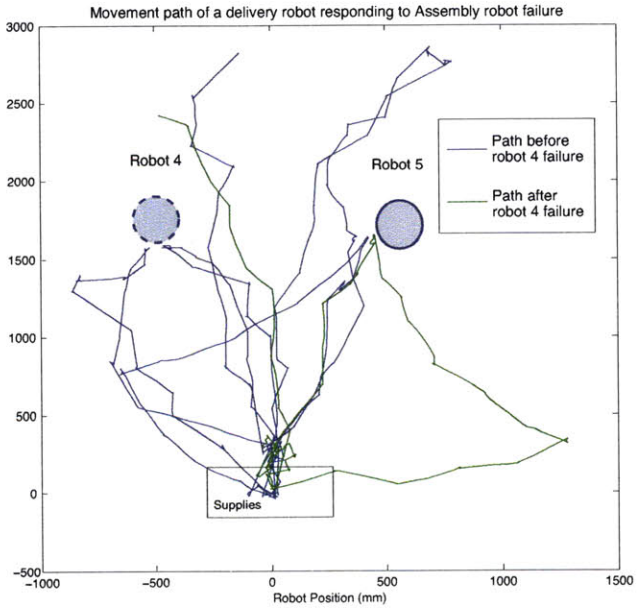


Figure 8-15: Adaptive behavior of the system: a delivery robot begins by delivering parts fairly to robot 4 and robot 5. When robot 4 (on the left) fails in the middle of the test, the delivery robot begins delivering only to robot 5.

Trial	**Runtime** (MM:SS)	**Avg. Runtime**	**Success**	**Failure**
1	06:05	06:05	1/1	
2	07:36	07:36	1/1	
3	07:20	07:20	1/1	
4	13:58	06:59	2/2	
5	37:33	06:16	6/6	
6	21:40	07:13	3/3	
7	14:18	04:46	3/3	
8	23:04	04:37	5/5	
9	41:28	06:55	6/6	
10	15:49	05:16	1/3	gripper weakened
11	71:05	05:55	11/12	dropped a part
12	23:17	04:39	5/5	
Total	04:43:13	06:54	45/48	

Table 8.3: Summary of Robot Delivery Test Runs

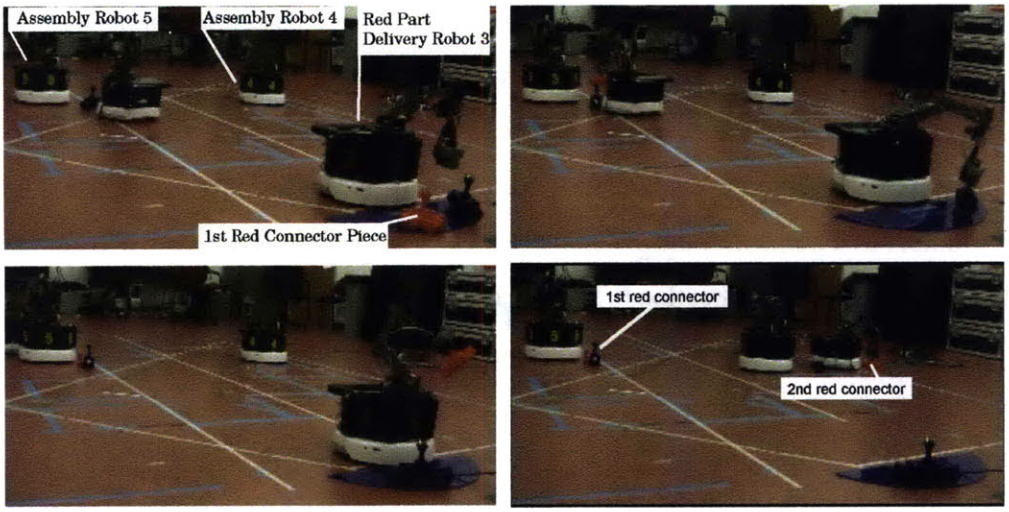


Figure 8-16: Snapshots of a test run of the even demanding mass delivery scenario. Assembly robots begin positioned at 2 different points of highest demand for parts. As the red connector parts are delivered, the maximum demanding mass for the entire map changes, causing the delivery robot to change delivery targets, first to robot 5, then to robot 4.

search for and pick up the correct type of part. This large amount of time caused a backup in the system: for all test runs in which both delivery robots ran at once, each delivery robot spent an average time of 2.57 minutes *per delivery* waiting for the other delivery robot to move out of the way. This is consistent with observations for the test runs in which only one delivery robot operated, showing an average round trip delivery time of only 6.90 minutes, an immediate 20% time decrease. This large chunk of time suggests further areas of research for the practical parallelization of the system.

8.3.3 Loose assembly

Preliminary tests of the assembly system add hand-offs of a part and placing down the part at the designated location to the delivery experiment.

Assembly Scenario

We use a 0.6m $\times$ 0.6m square blueprint which consists of four trusses on each side and four connectors at each corners. Figure 8-17 shows the blueprint on the GUI part of the snapshots, where the red squares corresponds to the connectors and the blue rectangles are

the trusses. For each location of the parts in the blueprint, we set a waypoint from which an assembly robot approaches to the final assembly location. The waypoints are chosen so that we can avoid the orientation problem due to the lack of degree-of-freedoms of the arm. At the start of construction, the blueprint is automatically divided into pieces that have the same number as a number of the assembly robots, according to the Voronoi partition which is decided by the starting location of the assembly robots.

The handoff started from a delivery continues by moving an assembly robot to the location the delivery robot has reported. Then, the assembly robot uses the search pattern in Algorithm 15 with just ± 10 deg instead of ± 90 deg. After picking up the delivered part, the assembly robot places down the part via the waypoint. Finally, it sends out a message containing the location of the placed part so that the other robot may set the assembled part as an obstacle. In the experiments, we use 0.2m $\times$ 0.2m square for the size of the obstacle which is booked in the grid map of each robot.

Assembly Experiments

We ran several tests with various combination of robots: 1 assembly and 1 delivery robots, 1 assembly and 2 delivery robots, and finally 2 assembly and 2 delivery robots.

Figure 8-17 and 8-18 shows snapshots of the experiment with 1 assembly robot and 2 delivery robots. The second and third pictures show handoff between the delivery robot 2 and the assembly robot 4. After a 40-minute test, the robots could locate all the parts at the designated locations. The loosely assembled structure had a few disorientations of the parts mainly because of the lack of the arm's dof. Even though the motion capture system provides up to millimeter accuracy, the mobile simply cannot achieve the precision and nor can the arm. Fortunately, the parts are designed such that they can self-align despite of the centimeter transitional and ten-degree rotational errors, we will try to assembly a tightly assembled square in the future.

Figure 8-19 shows trajectories of the three robots in the experiment. Clearly, the assembly robots drive around the parts on the blueprint, and we can speculate the waypoints by the wiggling trajectories at some points. A large amount of traffic is seen at the source cache (origin), and this indeed seriously slows down the execution.

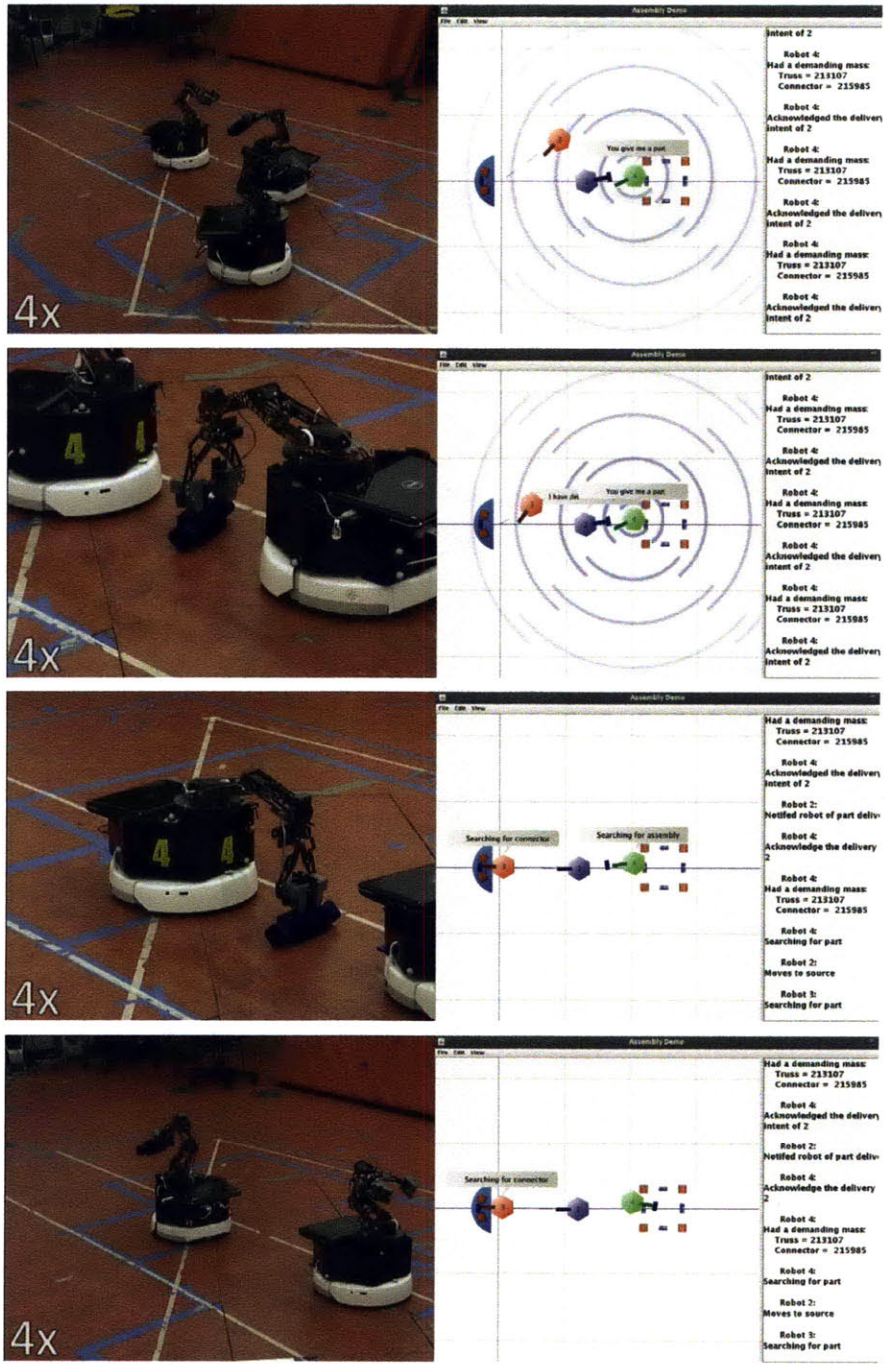


Figure 8-17: Snapshots of handoff and loose assembly.

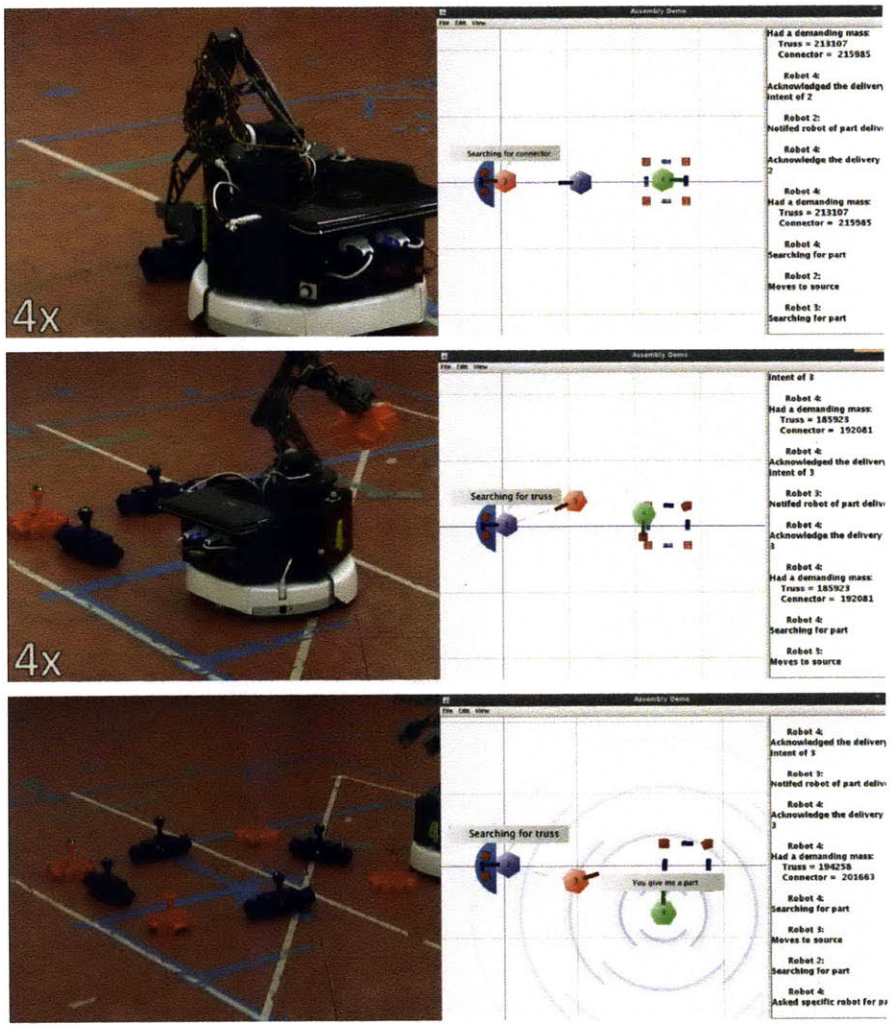


Figure 8-18: Snapshots of handoff and loose assembly.

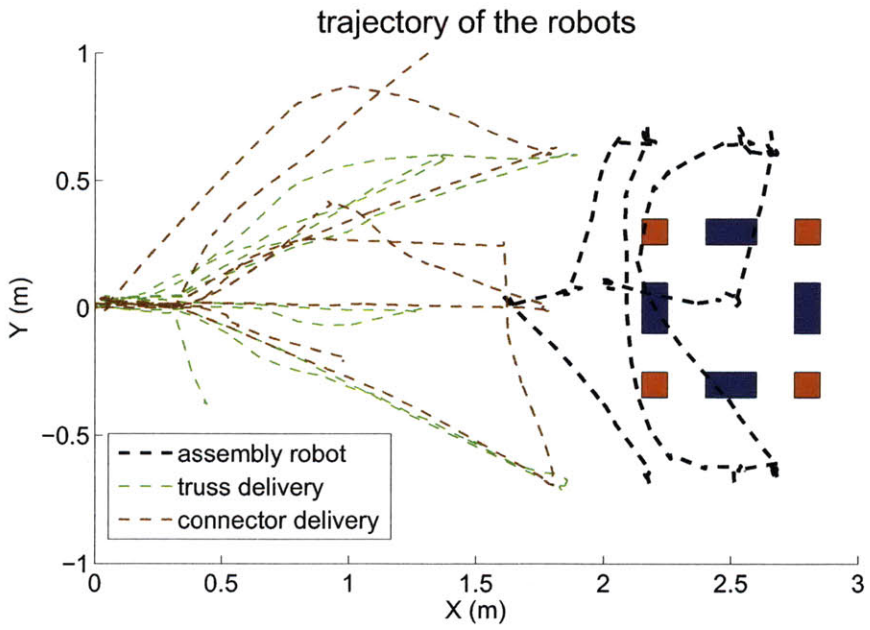


Figure 8-19: Trajectories of the robots. The blue rectangles and the red squares are trusses and connectors respectively. We can clearly see the assembly robot runs around the blueprint.

Robots Assembly/Delivery	**Runtime** (MM:SS)	**Avg. Handoff**	**Avg. Placement**	**Success**	**Failure**
1/1	40:08	2:15	1:08	8/8	
2/1	42:59	2:12	0:52	7/8	dropped a part
2/2	66:36	1:15	0:55	20/21	failed in grasping

Table 8.4: Summary of Robot Assembly Test Runs

The similar experiments with 2 assembly robots and 2 delivery robots have been done as shown in Figure 8-20 and 8-21. The experiment was sped up to about 30 minutes, since we used 1 more assembly robot and tuned the grasping algorithm. Figure 8-22 shows two sets of the trajectories the robots have made through the experiments.

The experimental results are summarized in Table 8.4. Note that each success includes delivery, handoff and placement. Therefore each assembly sequence is responsible for 4 manipulation sequences. The runtime is not much improved with multi-delivery robots since there is a bottleneck of picking up a part at the source cache.

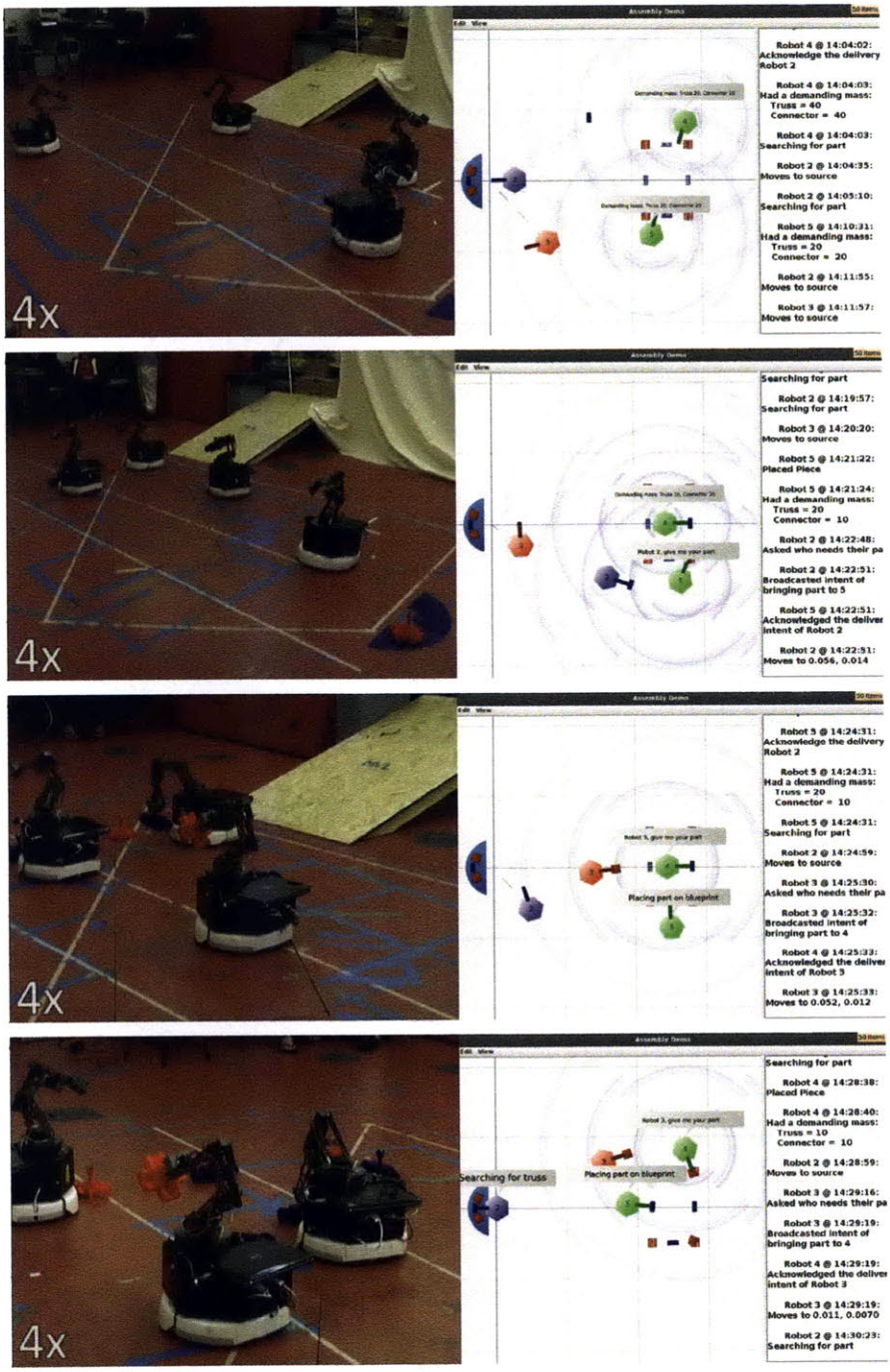


Figure 8-20: Snapshots of handoff and loose assembly by 2 delivery and 2 assembly robots.

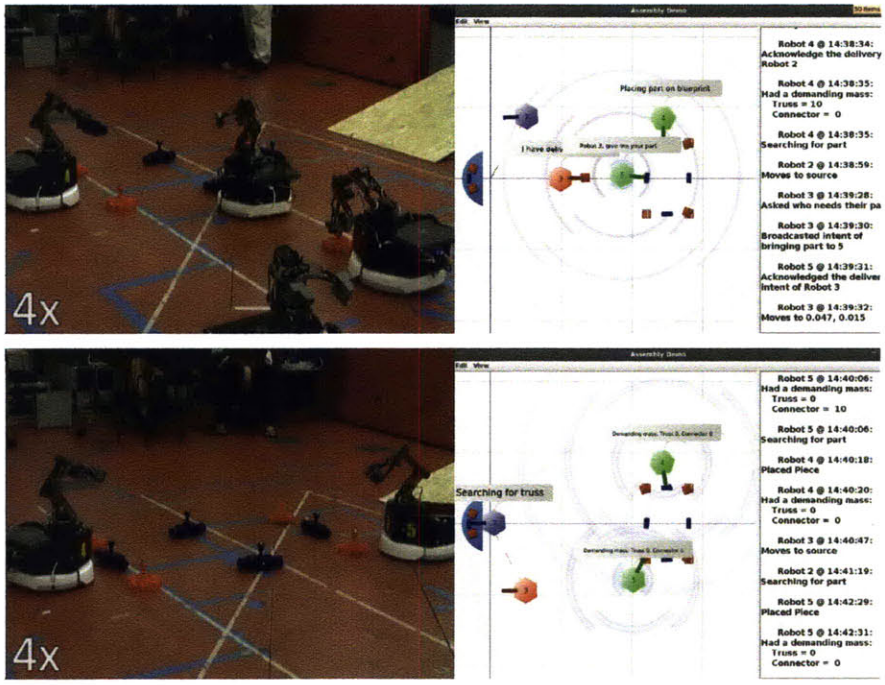


Figure 8-21: Snapshots of handoff and loose assembly by 2 delivery and 2 assembly robots.

8.3.4 Communication

UDP among robots

The delivery test runs of the robots requires that delivery robots communicate with assembly robots in order to confirm the delivery of a part to the assembly robots. When not engaged with a specific delivery robot making a delivery, each assembly robot broadcast a desire for a part delivery at a rate of 1Hz to alert any nearby delivery robot, so delivery robots trying to complete deliveries listened for assembly robots asking for a delivery and respond asynchronously. To complete the communication loop, we required the rebroadcasting and acknowledgement of all messages between 2 robots. Over 12 delivery test runs, on each round trip delivery, the delivery robot required 4.8 message packets total to deliver 2 messages to the target assembly robot, meaning each message from a delivery robot had to be resent at least once on average before a response was received from a target assembly robot. The assembly robots spend part of each delivery robot's delivery round asking for parts at a rate of 1Hz, meaning that during the average delivery, each assembly robot sent out an average of 180.5 messages before a delivery robot could pick up a part and respond

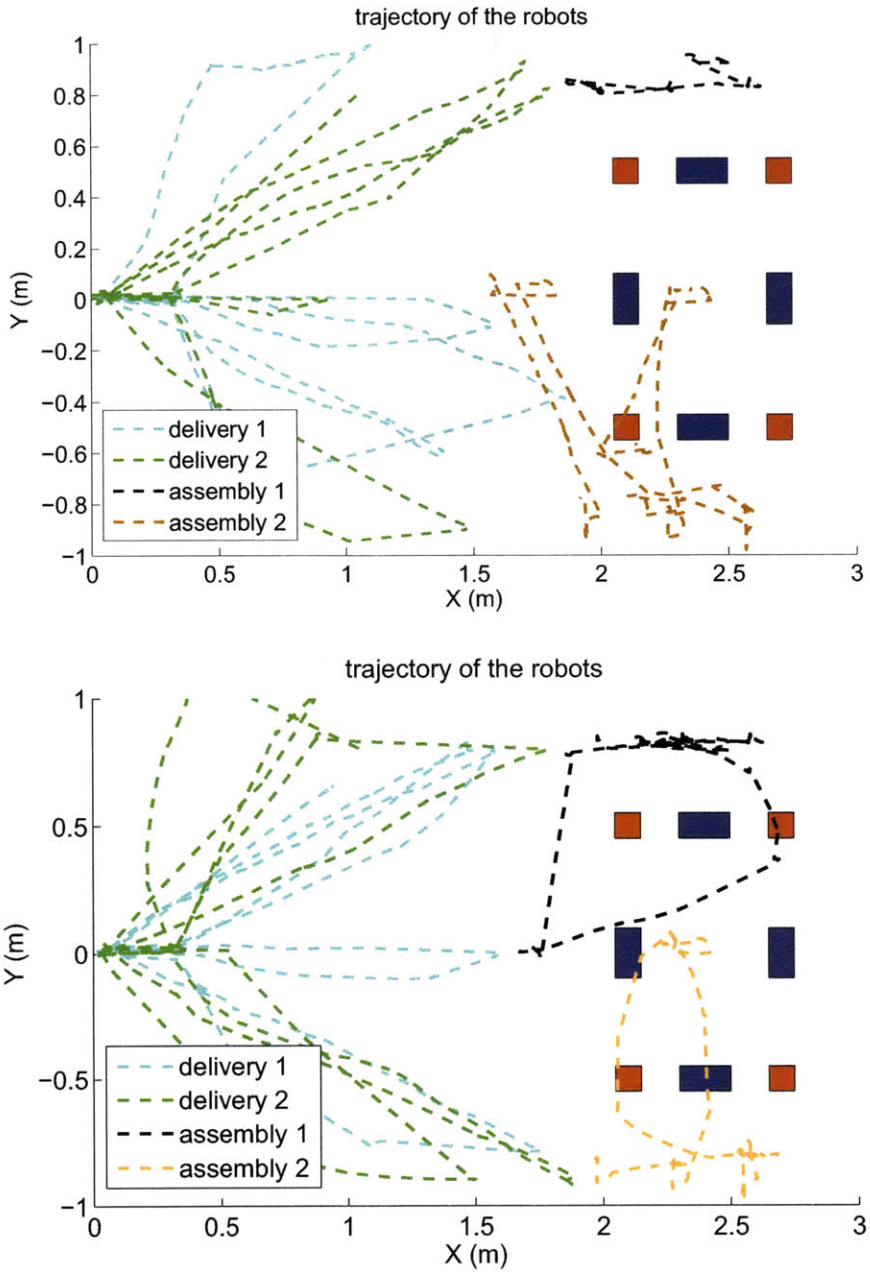


Figure 8-22: Trajectories of the four robots. The blue rectangles and the red squares are trusses and connectors respectively. In the upper figure, assembly robot 2 has more assigned parts than robot 3. In the lower figure, the assembly robots have the same amount of assigned parts.

to a needy assembly robot.

IR between a robot and a part

The smart parts used in this experiment broadcast data about what they are and their relative orientation to receivers mounted on the robot grippers. The parts also allow transmitters to modify a message portion of the data they broadcast. In over 1000 tests, robots were able to autonomously locate and grasp a part and modify its message with the part randomly placed in a semi-circular region with a 33cm with a 99.3% success rate, and 100% failure recovery.

Chapter 9

Conclusion and Future work

This thesis described a framework for distributed robotic construction where a team of networked robots which have specialized tasks (assembly and delivery of various parts) cover the target structure which is given by a density function, and perform their tasks with only local communication.

In order to divide the structure in equally-sized substructures, the equal-mass partitioning controller is introduced, guaranteeing convergence of the cost function that is the product of the all the masses. The algorithms construct Voronoi tessellations based on positions of the robots, and find the optimal next positions, and they work not only for a continuous domain but also for a discrete domain.

The graph partitioning algorithms enable mobile robots to cover a target graph with minimizing the cost functions. The target graph represents environments that can be inherently modeled by a set of nodes and edges or non-convex region. The two cost functions for locational optimization and equal-mass partitioning are used and the corresponding algorithms are designed based on vertex substitution. We show two-hop communication is required for convergence of the graph partitioning algorithms while a single hop suffices in a continuous domain.

An intuitive control criteria with probabilistic deployment and a gradient of the demanding masses is proposed to maintain a balance among the substructures. Implementation with two kinds of source materials (truss and connector) shows that the proposed algorithms assign an equal amount of construction work to the assembling robots, and ef-

fectively construct the target structures. We show the probabilistic delivery algorithm is an instance of a classic problem: balls into bins. Analysis leads to theoretical bounds for unbalance among the sub-structures that are empirically proven in simulation. Given the assumption that equal-mass partitioning has found the global optimum, the local search algorithm reduces the bound from $\sqrt{2\frac{m}{n}\log n}$ to $\frac{\log\log n}{\log d}$.

Based on the proposed approach, the algorithms are adaptive for several cases. For failure of robots, the convergence property is not affected by failure of delivering robots, and keeping equal-mass partitioning makes the system robust to failure of assembling robots. Construction with dynamic constraints is possible by incorporating the time-varying density function and corresponding controllers which is slightly modified from the original controller. Non-dependent source elements can be used by superposing density functions for the elements. Reconfiguration between two structures are implemented by substituting the target density function for difference between target density functions of two structures.

As for hardware implementation, we make a transition of a complex decentralized algorithm from theory to practice. The coordinated assembly by a multi robot system consists of four mobile manipulators and smart parts with the IR beacons to help communication between a robot and a part. In order to make the system demonstrate the desired algorithmic behavior, we combined the high-level algorithms controlling the actions of the robots with lower level controllers for viable communication channels, stable robot localization and navigation, collision avoidance, and part manipulation. The resulting system demonstrated a use for distributed robotics in industry that involved distributed control, autonomous and mobile robots, and an active ability to change their environment. Our next steps are focused on improving the capability of the assembly system, to demonstrate the use of the system for building of truss-like objects such as boxes and bookshelves.

9.1 Future direction and extension

9.1.1 Algorithm: partitioning

The proposed equal-mass partitioning algorithms divide a target structure into a number of subassemblies that equals the number of assembly robots. We prove the algorithms converge to local minima, however, we have not discussed the quality of the converged configuration. In an extreme case, even the global minimum may not exist when we use Voronoi tessellation. The power diagram can be a candidate to guarantee the existence of global minimum [85], however it also has a limit in distributed implementation.

Another thing we should consider is a range of communication, if we do not have enough assembly robots. Partitioning while maintaining connectivity may be challenging.

9.1.2 Hardware Implementation

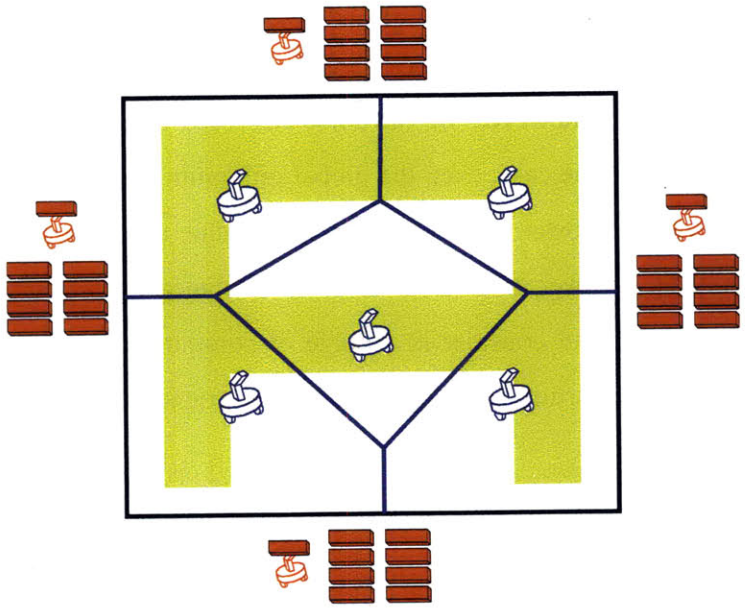
Better mobility and manipulability will be necessary for the next iteration of the hardware platform. The current system could show a flavor of assembly, however the tight assembly will require much more sensitive control of both navigation and manipulation. Now we only receive on/off signal from the parts, and capability of catching signal strength will help manipulation.

The battery life of smart parts should increase for a bigger scaled construction. The current battery lasts a couple of hours, and the electric circuits and the codes need to be optimized to reduce power consumption.

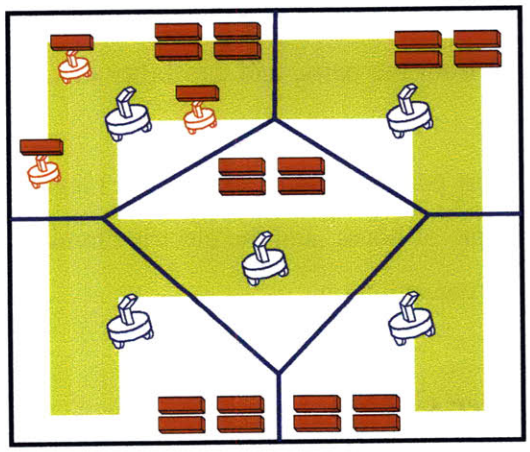
9.1.3 Multiple Source Caches

This thesis assumes only one location of the source cache, however, we may have multiple locations of the source caches which can be in the outside of the working area or scattered in the inside of the Voronoi regions as shown in Figure 9-1.

The theory can be extended to cover the multiple source locations. In either case, tracking of the remaining amount of source parts may be necessary for delivery robots. There is tradeoff between memory storage to track it and increasing travel distance due



(a)



(b)

Figure 9-1: Multiple source caches. They can be inside or outside of the working space.

to lack of the tracking information. For example, random walk of delivery robots will guarantee to find a source part after considerable amount of time whereas precise tracking of the remaining parts will lead to faster delivery. If the distribution of source parts $\mathcal{S}(\mathbf{q})$ is known, we can use the same probabilistic deployment algorithm; as picking up a location as $\mathbf{q} \sim \phi_t(\mathbf{q})$ in Algorithm 5, a delivery robot tries to find a source part by moving to a randomly selected location based on $\mathcal{S}(\mathbf{q})$ in the IDLE state.

Another option is that delivery robots may learn the distribution as in [101] if they are able to sense the source parts. Each delivery robot initializes its distribution, and the consensus algorithm which broadcasts any change of $\mathcal{S}(\mathbf{q})$ via the mesh network will update the distribution of all the delivery robots in real time during delivery until the robots finalize the distribution. The robots can also pick a random location given the *current* distribution.

Source caches inside the regions When source parts are scattered in the Voronoi regions as shown in Figure 9-1(b), assembly robots can work as network hubs for delivery robots inside their Voronoi partitions so that they track the remaining amount of source and help delivery. Algorithm 5 can be modified so that the delivery robots skip the ToSOURCE state and directly go into a Voronoi partition by running the ToTARGET state as if they already picked a part. The assembly robot in the partition communicates with the delivery robot to have the robot pick up a source part if any in the partition or send it to neighboring partitions if there is no part in the partition.

In other way, the assembly robots may run the consensus algorithm to get the distribution of source parts before delivery starts, and let the delivery robot know the distribution when they come into the Voronoi partitions. Once the delivery robots have received the distribution, they can use the same algorithm for multiple source caches.

9.1.4 Scheduling algorithm for delivery

The delivery based on the probabilistic deployment algorithm and the local search algorithm is intuitive and effective for uniform delivery. We may extend the algorithms for faster delivery while keeping the uniformity by introducing a scheduling algorithm which considers parameters such as (1) the distance between a robot location and a selected loca-

tion from the probabilistic deployment algorithm, (2) speed of delivery robots, and (3) the difference of demanding masses of the neighbor assembly robots. Particularly if we have the multiple locations of source caches, the scheduling algorithm may want to send a delivery robot to nearby assembly robots in order to save the delivery time, by assigning priority based on the distance from the delivery robot. Also, ignoring small difference in demanding masses may speed up construction, which may arise a little unbalance. For example, if a delivery robot picks up a part in the left source cache in Figure 9-1(a), it may want to select an assembly robot nearby the left side rather than the right side because this biased choice will save the travel distance. We have to have a cost function for the scheduling algorithm so that we optimize it with the three parameters.

A scheduling algorithm has been extensively considered in computer science literature, and we can customize the off-the-shelf algorithm for construction.

9.2 Lesson learned

The theory and practice work in this thesis has taught us several important lessons.

Algorithm: partitioning The equal-mass partitioning algorithms work well most of time, however undesirable local minima was inevitable time to time. Guaranteeing the global optimum has been the ultimate goal for distributed coverage, and our algorithms are also in the same boat.

Algorithm: uniform delivery We have proposed the delivery algorithm based on probabilistic deployment and local search, and performance has been analyzed using the balls into bins problem. We have learned the local search algorithm needs to be tuned for adaptation because the algorithm always drives a delivery robot to a robot with the maximum demanding mass, which works perfectly in a static case where the Voronoi partitions are fixed during construction. However, when the partitions change during construction, the algorithm may lead to unbalanced assembly work since the adaptation algorithm tries to balance masses during assembly and this leads to change of the demanding mass. A suit-

able scheduling algorithm will solve this problem.

Hardware Implementation The transition from theory to practice is always challenging, and our system is not an exception.

Our main challenge was to keep the hardware operational. We have an inexpensive platform and our platform was not very reliable. The Roomba iCreate mobile base does not provide good odometry, therefore we had to rely solely on the external motion capture system, which led to non-smooth navigation. A mobile base with encoders should be used in the next iteration for better mobility. The arm has only 4-dof and we could not locate parts in the right orientation even on 2D, which limits tight assembly. One more rotational degree of freedom at the end of the gripper will suffice for our 2D assembly. Communication between the instrumented gripper and the smart parts greatly enhance grasping.

Bibliography

[1] A. Scott Howe and Ian Gibson. Trigon robotic pairs. In *AIAA Space 2006 Conference*, 2006.

[2] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[3] J. Ashayeri, R. Heuts, and B. Tammel. A modified simple heuristic for the p-median problem, with facilities design applications. *Robotics and Computer-Integrated Manufacturing*, 21(4-5):451 – 464, 2005.

[4] Pasquale Avella, Maurizio Boccia, Antonio Sforza, and Igor Vasil'Ev. A branch-and-cut algorithm for the median-path problem. *Comput. Optim. Appl.*, 32(3):215–230, 2005.

[5] Pasquale Avella, Maurizio Boccia, Antonio Sforza, and Igor Vasil'Ev. An effective heuristic for large-scale capacitated facility location problems. *Journal of Heuristics*, 15(6):597–615, 2009.

[6] Pasquale Avella, Antonio Sassano, and Igor Vasil'ev. Computational study of large-scale p-median problems. *Math. Program.*, 109(1):89–114, 2007.

[7] Nora Ayanian and Vijay Kumar. Decentralized feedback controllers for multi-agent teams in environments with obstacles. In *IEEE International Conference on Robotics and Automation*, "Pasadena, May 2008.

[8] Stephen Barnard. Pmrsb: Parallel multilevel recursive spectral bisection. In *In Supercomputing*, 1995.

[9] Opher Baron, Oded Berman, Dmitry Krass, and Qian Wang. The equitable location problem on the plane. *European Journal of Operational Research*, 183(2):578–590, December 2007.

[10] Roberto Battiti, Alan Bertossi, and Anna Cappelletti. Multilevel reactive tabu search for graph partitioning. Technical report, 1999.

[11] J. E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21(2):270–273, August 1985.

[12] J. E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[13] C. Beltran, C. Tadonki, and J. Ph. Vial. Solving the p-median problem with a semi-lagrangian relaxation. *Comput. Optim. Appl.*, 35(2):239–260, 2006.

[14] Ben Iannotta. Creating robots for space repairs. *Aerospace America*, pages 36–40, May 2005.

[15] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Comput.*, 36(5):570–580, 1987.

[16] Dimitris Bertsimas and Garrett. Van Ryzin. Stochastic and dynamic vehicle routing in the euclidean plane with multiple capacitated vehicles. Technical report, 1991.

[17] Adrienne Bolger. Coordinated part delivery using distributed planning. Master's thesis, Massachusetts Institute of Technology, CSAIL Distributed Robotics Laboratory, 2010.

[18] Carlos H. Caicedo-Nunez and Milos Zefran. A coverage algorithm for a class of non-convex regions. In *CDC*, pages 4244–4249, 2008.

[19] Carrick Detweiler, Marsette Vona, Yeoreum Yoon, Seung-kook Yun, and Daniela Rus. Self-assembling mobile linkages. *IEEE Robotics and Automation Magazine*, 14(4):45–55, 2007.

[20] Cem Unsal, Han Kiliccote, and Pradeep Khosla. A modular self-reconfigurable bipartite robotic system: Implementation and motion planning. *Autonomous Robots*, 10(1):23–40, January 2001.

[21] Hyeun-Seok Choi, Chang-Soo Han, Kye young Lee, and Sang heon Lee. Development of hybrid robot for construction works with pneumatic actuator. *Automation in Construction*, 4, August 2005.

[22] The Connection Machine Cm-System, Zdenek Johan, Zdenek Johan, Kapil K. Mathur, Kapil K. Mathur, S. Lennart Johnsson, S. Lennart Johnsson, Thomas J. R. Hughes, and Thomas J. R. Hughes. An efficient communication strategy for finite element methods on the connection machine cm-5 system. In *Methods on the Connection Machine CM–5 system. Computer Methods in Applied Mechanics and Engineering*, pages 11–3.

[23] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. 20(2):243–255, 2004.

[24] Daniela Rus and Marsette Vona. Self-reconfiguration planning with compressible unit modules. In *IEEE International Conference on Robotics and Automation*, 1999.

[25] Daniela Rus and Marsette Vona. A basis for self-reconfiguring robots using crystal modules. In *Proceedings of the IEEE/RSJ International Conference on Intellilegent Robots and Systems*, pages 2194–2202, October 2000.

[26] Daniela Rus and Marsette Vona. A physical implementation of the self-reconfiguring crystalline robot. In *IEEE International Conference on Robotics and Automation*, pages 1726–1733, 2000.

[27] Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, January 2001.

[28] David G. Duff, Mark Yim, and Kimon Roufas. Evolution of polybot: A modular reconfigurable robot. In *Proceedings of the Harmonic Drive International Symposium*, Nagano, Japan, November 2001.

[29] J. W. Durham, R. Carli, P. Frasca, and F. Bullo. Discrete partitioning and coverage control with gossip communication. In *In ASME Dynamic Systems and Control Conference*, Hollywood, CA, Oct 2009.

[30] Martin Erwig and Fernuniversitat Hagen. The graph voronoi diagram with applications. *Networks*, 36:156–163, 2000.

[31] Scott E. Fahlman. A planning system for robot construction tasks. Technical report, Cambridge, MA, USA, 1973.

[32] Charbel Farhat and Michel Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *International Journal for Numerical Methods in Engineering*, 36(5):745–764, 1993.

[33] Matthew Faulkner. Instrumented tools and objects: Design,algorithms, and applications to assembly tasks. Master's thesis, Massachusetts Institute of Technology, CSAIL Distributed Robotics Laboratory, June-August 2009.

[34] E F Fichter. A stewart-platform based manipulator: general theory and practical construction. *International Journal of Robotics Research*, 5(2):157–182, 1986.

[35] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th Design Automation Conference*, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[36] Per-Olof Fjallstrom. Algorithms for graph partitioning: A survey, 1998.

[37] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

[38] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):77–98, 2001.

[39] A. Ganguli, J. Cortes, and F. Bullo. Distributed deployment of asynchronous guards in art galleries. pages 1416–1421, Minneapolis, MN, June 2006.

[40] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979.

[41] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, New York, NY, USA, 1974. ACM.

[42] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[43] F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[44] F. Glover. Tabu search - part ii. *ORSA Journal on Computing*, 2:4–32, 1990.

[45] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[46] Gregory J. Hamlin and Arthur C. Sanderson. Tetrobot: A modular approach to parallel robotics. *IEEE Robotics & Automation Magazine*, pages 42–49, March 1997.

[47] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. In *Journal of Algorithms*, pages 649–657, 1998.

[48] Henrik Hautop Lund, Richard Beck, and Lars Dalgaard. ATRON hardware modules for self-reconfigurable robotics. In Sugisaka and Takaga, editor, *Proceedings of 10th International Symposium on Artificial Life and Robotics (AROB'10), ISAROB*, Oita, 2005.

[49] Hisanori Amano, Koichi Osuka, and Tzyh-Jong Tarn. Development of vertically moving robot with gripping handrails for fire fighting. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 661–667, Maui, HI, 2001.

[50] Horst D. Simon I, Horst D. Simon, and Horst D. Simon. Partitioning of unstructured problems for parallel processing, 1991.

[51] Jacob Everist, Kasra Mogharei, Harshit Suri, Nadeesha Ranasinghe, Berok Khoshnevis, Peter Will, and Wei-Min Shen. A system for in-space assembly. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2356–2361, Sendai, Japan, 2004.

[52] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):539–560, 1979.

[53] Kasper Støy. The ATRON self-reconfigurable robot: challenges and future directions. Presentation at the Workshop on Self-reconfigurable Robotics at the Robotics Science and Systems Conference, July 2005.

[54] Keith D. Kotay and Daniela L. Rus. Navigating 3d steel web structures with an inchworm robot. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 1996.

[55] Keith Kotay. *Self-Reconfiguring Robots: Designs, Algorithms, and Applications.* PhD thesis, Dartmouth College, December 2003.

[56] Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray. The self-reconfiguring robotic molecule. In *IEEE International Conference on Robotics and Automation*, 1998.

[57] Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray. The self-reconfiguring robotic molecule: Design and control algorithms. In *Workshop on the Algorithmic Foundations of Robotics*, 1998.

[58] Krishnaram Kenthapadi and Rina Panigrahy. Balanced allocation on graphs. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 434–443, New York, NY, USA, 2006. ACM.

[59] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[60] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[61] Seung kook Yun, David Alan Hjelle, Hod Lipson, and Daniela Rus. Planning the reconfiguration of grounded truss structures with truss climbing robots that carry truss elements. In *Proc. of IEEE/RSJ IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009.

[62] Seung kook Yun and Daniela Rus. Optimal distributed planning of multi-robot placement on a 3d truss. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, USA, October 2007.

[63] Seung kook Yun and Daniela Rus. Optimal distributed planning for self assembly of modular manipulators. In *Proc. of IEEE/RSJ IEEE International Conference on Intelligent Robots and Systems*, pages 1346–1352, Nice, France, Sep 2008.

[64] Seung kook Yun and Daniela Rus. Self assembly of modular manipulators with active and passive modules. In *Proc. of IEEE/RSJ IEEE International Conference on Robotics and Automation*, pages 1477–1482, May 2008.

[65] Seung kook Yun, Mac Schwager, and Daniela Rus. Coordinating construction of truss structures using distributed equal-mass partitioning. In *Proc. of the 14th International Symposium on Robotics Research*, Lucern, Switzerland, August 2009.

[66] Seung kook Yun, Yeoreum Yoon, and Daniela Rus. Self assembly of modular manipulators with active and passive modules. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems Workshop on Self-Reconfigurable Robots*, San Diego, USA, October 2007.

[67] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. In *In Proceeding of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, 1998.

[68] A. A. Kuehn and M. J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9(4):643–666, 1963.

[69] Seung Yeol Lee, Kye Young Lee, Sang Heon Lee, Jin Woo Kim, and Chang Soo Han. Human-robot cooperation control for installing heavy construction materials. *Auton. Robots*, 22(3):305–319, 2007.

[70] R. Leland and B. Hendrickson. An empirical study of static load balancing algorithms. In *Performance Comput. Conf.*, pages 682–685, 1994.

[71] Robert Leland and Bruce Hendrickson. An empirical study of static load balancing algorithms. In *In Proc. Scalable High-Performance Comput. Conf.*, pages 682–685, 1994.

[72] Abilio Lucena and John E. Beasley. Branch and cut algorithms. pages 187–221, 1996.

[73] M. Almonacid, R. J. Saltarén, R. Aracil, and O. Reinoso. Motion planning of a climbing parallel robot. *IEEE Transactions on Robotics and Automation*, 19(3):485–489, 2003.

[74] Harpal Maini, Kishan Mehrotra, Chilukuri Mohan, and Sanjay Ranka. Genetic algorithms for graph partitioning and incremental graph partitioning. In *Supercomputing '94: Proceedings of the 1994 conference on Supercomputing*, pages 449–457, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[75] F. E. Maranzana. On the location of supply points to minimize transportation costs. *IBM Syst. J.*, 2(2):129–135, 1963.

[76] Loic Matthey, Spring Berman, and Vijay Kumar. Stochastic strategies for a swarm robotic assembly system. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1953–1958. IEEE, 2009.

[77] Michael Nechyba and Yangsheng Xu. SM2 for new space station structure: Autonomous locomotion and teleoperation control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1765–1770, May 1994.

[78] Michael Nechyba and Yangsheng Xu. Human-robot cooperation in space: SM2 for new space station structure. *IEEE Robotics and Automation Magazine*, 2(4):4–11, December 1995.

[79] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Automatic mesh partitioning. pages 57–84, 1993. Vol 56.

[80] Michael David Mitzenmacher, Michael David Mitzenmacher, and Michael David Mitzenmacher. The power of two choices in randomized load balancing. Technical report, IEEE Transactions on Parallel and Distributed Systems, 1996.

[81] A D Nease and E F Alexander. Air force construction automation/robotics. *Automation in Construction*, 3, May 1994.

[82] MC Nechyba and Y. Xu. Human-robot cooperation in space: SM^2 for new spacestation structure. *Robotics & Automation Magazine, IEEE*, 2(4):4–11, 1995.

[83] C. Ozturan, H. L. deCougny, M. S. Shephard, and J. E. Flaherty. Parallel adaptive mesh refinement and redistribution on distributed memory computers. Technical report, Comput. Methods Appl. Mech. Engrg, 1993.

[84] Chris Parker, Hong Zhang, and Ronald Kube. Blind bulldozing: Multiple robot nest construction. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, USA, October 2003.

[85] Marco Pavone, Emilio Frazzoli, and Francesco Bullo. Distributed algorithms for equitable partitioning policies: Theory and applications. In *IEEE Conference on Decision and Control*, Cancun, Mexico, Dec 2008.

[86] Peter J. Staritz, Sarjoun Skaff, Chris Urmson, and William Whittaker. Skyworker: A robot for assembly, inspection and maintenance of large scale orbital facilities. In *IEEE ICRA*, pages 4180–4185, Seoul, Korea, 2001.

[87] L. C. A. Pimenta, M. Schwager, Q. Lindsey, V. Kumar, D. Rus, R. C. Mesquita, and G. A. S. Pereira. Simultaneous coverage and tracking (scat) of moving targets with robot networks. In *Proceedings of the Eighth International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Guanajuato, Mexico, December 2008.

[88] Luciano C. A. Pimenta, Vijay Kumar, Renato C. Mesquita, and Guilherme A. S. Pereira. Sensing and coverage for a network of heterogeneous robots. In *CDC*, pages 3947–3952, 2008.

[89] Alex Pothen, Horst D. Simon, and Kan-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[90] Martin Raab and Angelika Steger. Balls into bins - a simple and tight analysis, 1998.

[91] J. Reese. Solution methods for the p-median problem: An annotated bibliography. *Networks*, 48:125–142, 2006.

[92] C. Revelle and R. Swain. central facilities location. *Geographical Analysis*, 2:30–42, 1970.

[93] Robert L. Williams II and James B. Mayhew IV. Cartesian control of VGT manipulators applied to DOE hardware. In *Proceedings of the Fifth National Conference on Applied Mechanisms and Robotics*, Cincinnati, OH, October 1997.

[94] K. E. Rosing, C. S. Revelle, and H. Rosing-Vogelaar. The p-median and its linear programming relaxation: An approach to large problems. *The Journal of the Operational Research Society*, 30(9):815–823, 1979.

[95] Youssef G. Saab and Vasant B. Rao. Stochastic evolution: a fast effective heuristic for some generic layout problems. In *DAC '90: Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 26–31, New York, NY, USA, 1990. ACM.

[96] Satoshi Murata, Haruhisa Kurokawa, Eiichi Yoshida, Kohji Tomita, and Shigeru Kokaji. A 3-d self-reconfigurable structure. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 432–439, Leeuven, Belgium, May 1998.

[97] Crystal Schuil, Matthew Valente, Justin Werfel, and Radhika Nagpal. Collective construction using lego robots. In *AAAI'06: proceedings of the 21st national conference on Artificial intelligence*, pages 1976–1977. AAAI Press, 2006.

[98] M. Schwager, B. Julian, and D. Rus. Optimal coverage for multiple hovering robots with downward-facing cameras. In *In the Proceedings of the International Conference on Robotics and Automation (ICRA 09), Accepted*, Kobe, Japan, May 2009.

[99] M. Schwager, J. McLurkin, J. J. E. Slotine, and D. Rus. From theory to practice: Distributed coverage control experiments with groups of robots. In *Proceedings of International Symposium on Experimental Robotics*, Athens, Greece, July 2008.

[100] Mac Schwager, Daniela Rus, and Jean-Jacques E. Slotine. Decentralized, adaptive control for coverage with networked robots. *International Journal of Robotics Research*, 28(3):357–375, March 2009.

[101] Mac Schwager, Jean-Jacques E. Slotine, and Daniela Rus. Decentralized, adaptive control for coverage with networked robots. In *Proceedings of International Conference on Robotics and Automation*, Rome, April 2007.

[102] Eric Schweikardt and Mark D. Gross. roblocks: a robotic construction kit for mathematics and science education. In *ICMI '06: Proceedings of the 8th international conference on Multimodal interfaces*, pages 72–75, New York, NY, USA, 2006. ACM.

[103] Seung-kookYun and Daniela Rus. Adaptation to robot failures and shape change in decentralized construction. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2010.

[104] David B. Shmoys, E'va Tardos, Eva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *In Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[105] S. Skaff, P. Staritz, and WL Whittaker. Skyworker: Robotics for space assembly, inspection and maintenance. *Space Studies Institute Conference*, 2001.

[106] Ashley Stroupe, Terry Huntsberger, Avi Okon, Hrand Aghazarian, and Matthew Robinson. Behavior-based multi-robot collaboration for autonomous construction tasks. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Alberta, Canada, August 2005.

[107] Michael B. Teitz and Polly Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *OPERATIONS RESEARCH*, 16:955–961, 1968.

[108] A. Vidwans, Y. Kallinderis, and V. Venkatakrishnan. A parallel dynamic load balancing algorithm for 3-d adaptive unstructured grids. *AIAA Journal*, 32:497–505, 1993.

[109] Justin Werfel. *Anthills built to order: automating construction with artificial swarms*. PhD thesis, Cambridge, MA, USA, 2006. Adviser-Nagpal, Radhika and Adviser-Seung, H. Sebastian.

[110] Justin Werfel. Robot search in 3d swarm construction. In *SASO '07: Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems*, pages 363–366, Washington, DC, USA, 2007. IEEE Computer Society.

[111] Justin Werfel and Radhika Nagpal. Extended stigmergy in collective construction. *IEEE Intelligent Systems*, 21(2):20–28, 2006.

[112] Justin Werfel and Radhika Nagpal. International journal of robotics research. *Three-dimensional construction with mobile robots and modular blocks*, 3-4(27):463–479, 2008.

[113] William Doggett. Robotic assembly of truss structures for space systems and future research plans. In *IEEE Aerospace Conference Proceedings*, March 2002.

[114] Roy D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Pract. Exper.*, 3(5):457–481, 1991.

[115] H. Yamada, H. Kato, and T. Muto. Master-slave control for construction robot teleoperation. *Journal of Robotics and Mechatronics*, 15:54–60, 2003.

[116] H. Yamada and T. Muto. Construction tele-robotic system with virtual reality (cg presentation of virtual robot and task object using stereo vision system). *Control Intell. Syst.*, 35(3):195–201, 2007.

[117] H. Yamada, T. Muto, and G. Ohashi. Development of a telerobotics system for construction robot using virtual reality. In *Proc. European Control Conf.*, Germany, 1999.

[118] Zaidi Mohd Ripin, Tan Beng Soon, A.B. Abdullah, and Zahurin Samad. Development of a low-cost modular pole climbing robot. In *TENCON*, volume I, pages 196–200, Kula Lumpur, Malaysia, 2000.

[119] D. Zhao, Y. Xia, H. Yamada, and T. Muto. Presentation of realistic motion to the operator in operating a tele-operated construction robot. *Journal of Robotics and Mechatronics*, 14:98–104, 2002.

[120] D. Zhao, Y. Xia, H. Yamada, and T. Muto. Control method for realistic motions in a construction tele-robotic system with a 3-dof parallel mechanism. *Journal of Robotics and Mechatronics*, 15:361–368, 2003.