Constraint-Aware Coordinated Robotic Construction of Generic Structures

by

David Benjamin Stein

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

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2011

© David Benjamin Stein, MMXI. All rights reserved.

The author hereby grants to MIT and MIT Lincoln Laboratory permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author . . . . . .
Department of Electrical Engineering and Computer Science
June 20, 2011

Certified by . . . . . .
Daniela Rus
Professor
Thesis Supervisor

Accepted by . . . . . .
Christopher Terman
Chairman, Department Committee on Graduate Theses

Constraint-Aware Coordinated Robotic Construction of Generic Structures

by

David Benjamin Stein

Submitted to the Department of Electrical Engineering and Computer Science
on June 20, 2011, in partial fulfillment of the
requirements for the degree of
Master of Engineering

Abstract

This thesis considers the problem of distributed construction. We focus on the problem of designing algorithms for robots working on large stationary construction projects. We present fully distributed algorithms which tend to maximize the amount of work being done by the system given that robots may be added or removed from a site in real time. We also present experimental implementations and analysis of our both algorithms and the foundational work this thesis extends.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgements

Thanks to the Boeing Corporation for sponsoring this research; to Daniela Rus for her advice and guidance over the past two years; to the people who worked on this project with me: Seungkook Yun, Matthew Faulkner, Lauren White, Adrienne Bolger, and T. Ryan Schoen; to Brian Julian, Marek Doniec and the members of the Distributed Robotcs Laboratory who helped teach me the ropes, and of course to my family and friends for their support.

Finally, an additional thank you to Ryan and Daniela for encouraging me to complete this thesis after nearly two years of leave from academia.

Contents

List of Figures

List of Tables

Organization of Thesis

This thesis is arranged in order of motivation, but each chapter is intended to be readable given only the problem formulation (ch. 3). A high level overview of the problem space and the algorithmic structure this work uses as a foundation is introduced in chapter 3 with an experimental analysis of other work based on that foundation provided in chapter 4. The main algorithmic contribution of this thesis is covered in chapter 5. The experimental validation of these algorithms is covered in chapters 6 and 7.

Chapter 1

Introduction

There has been a clear motivation to find mechanical and robotic solutions to construction tasks, which can require work that is dangerous, difficult, or unpleasant for humans to perform. Human workers also can take longer to complete dedicated tasks than their robotic counterparts. These factors have lead to much research in robotic assembly lines, which now handle not only mass production, but even to-order, customized, and low-volume fabrication [3, 4].

Despite this, current state of the art for construction of large and static structures that do not fit into a traditional assembly line approach still rely heavily on humans performing tasks that could be potentially automated. This leads to workers performing potentially avoidable tasks that are dangerous or unpleasant. In 2009, 14% of workplace fatalities in the US occurred to construction workings, making construction the occupation that experienced the second most deaths that year and providing a clear motivation to develop automated solutions to construction tasks (Figure 1-1)[5].

Advances in robotic manipulation, mobility, and localization have reached a point where providing mobility in constrained dynamic environments is feasible for service robots. These advances will facilitate a discussion of largely automated construction under a new paradigm of unmoving structures being worked on by mobile robots. However, this approach requires the robots to have local intelligence, and either depend on a central controller - a single point of failure for the entire system - or work towards a goal as a fully

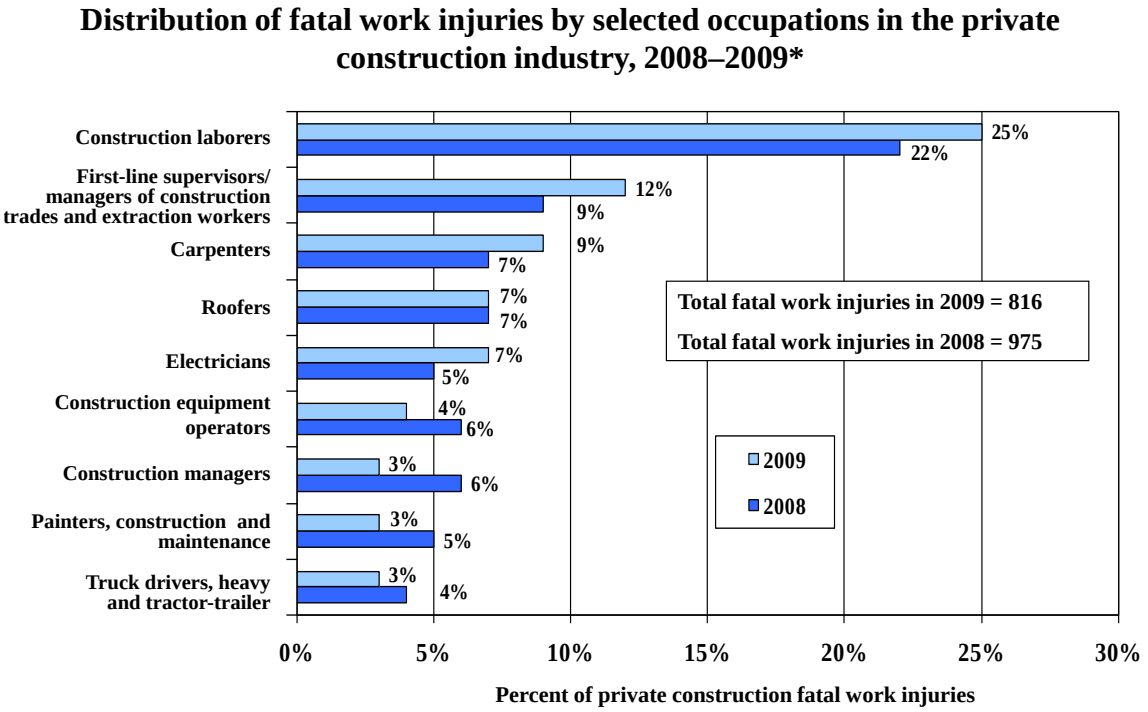


Figure 1-1: Construction work is dangerous, especially general construction. We would like to remove the need for human performance of the most dangerous and unpleasant tasks. Figure reproduced from a 2010 press release from the U.S. Bureau of Labor Statistics.

or partially distributed system. We focus on developing a fully autonomous distributed construction system.

1.1 Distributed Construction

We consider the problem of distributed construction. If we imagine a construction task with two major classes of problems – carrying parts to the site, which requires less high-precision actuation but a high load capacity (i.e. carrying a plank to the work site); and connecting parts to a structure, which places more emphasis on precision and sensing (i.e. screwing the plank in place) – we can see that there is a tendency towards creating a heterogeneous system. We focus on the problem of designing algorithms that tend to maximize the amount of work being done in a system given delivery and assembly robots which may be further sub-specialized, and which can be added or removed from a site in real time in such a way that the system reacts. We also dismiss using a single head node, as that introduces limits both in fault tolerance and in scalability due to communication.

1.2 Contributions

This work is founded strongly on the framework described in [6], which outlines a general approach distributed assembly of truss structures. This thesis extends that work in a number of ways.

1.2.1 Experimental Analysis of [1]

We performed experimental analysis of the algorithms described in [1]. Limiting classes of robots to assembly and delivery and shrinking communication radii in software, we implemented the distributed controllers described by Yun et. al. and identified key areas for improvement and modification. We also developed a low-cost platform for performing future experiments on distributed construction, and describe an alternative to vision or range

sensing for object recognition and manipulation that moves some intelligence into the part being manipulated. This is discussed at length in Chapter 4

1.2.2 Part Ordering in Assembly Tasks

In section 5.3 we describe an algorithm for deciding which part to place within a partition such that key global invariants are maintained. In this thesis we consider the reachability of all active work sites and the physical stability of a partially constructed system. We demonstrate that even in a dynamic system this priority can be computed efficiently using only local information.

1.2.3 Discrete Partitioning

The previous work in partitioning assumes either some continuous differentiable function describing the importance of any location in a space or a graph connecting points. In section 5.2 we demonstrate an algorithm for partitioning sets of points in Cartesian space dynamically.

1.2.4 Experiment for Coordinated Construction with Part Ordering

Using the next iteration of the platform introduced in chapter 6, we provide experimental validation that our new controllers and extensions of the algorithms from [6] address the areas of interest we identified in our initial experiments.

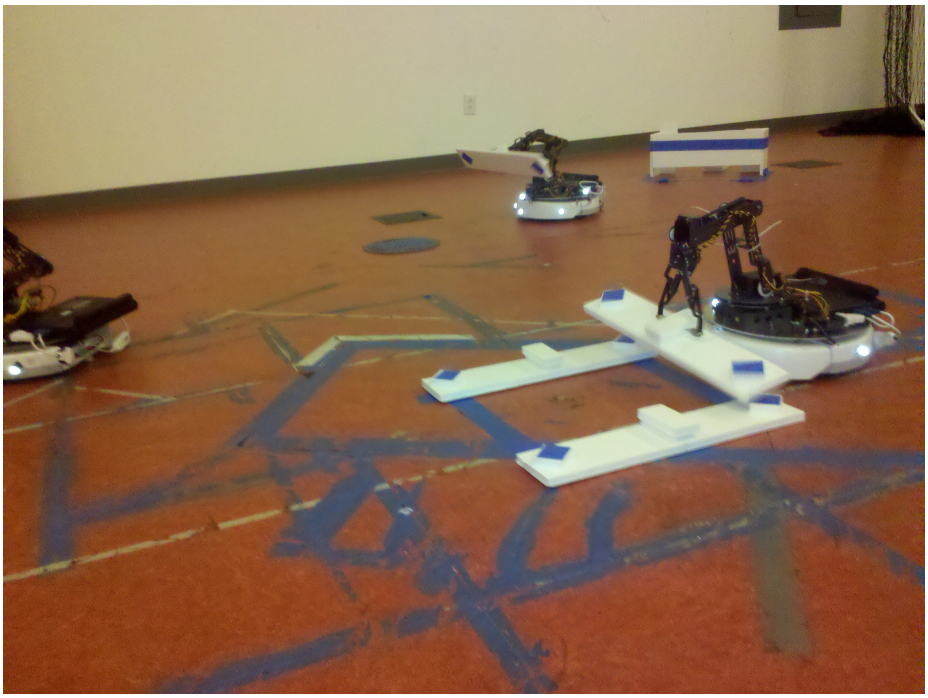


Figure 1-2: Image from experimental implementation the algorithms presented in this thesis. A full discussion of our implementation can be found in chapter 6

Chapter 2

Related Work

This research addresses problems in distributed control and task partitioning, which have been explored both in the context of robots, and in several related fields. We also look to examples from industry and current practice in construction as inspiration and justification for our models and approaches.

2.1 Construction Robots

Our problem definition, presented in detail in Chapter 3, is inspired by current industry models. Automated robotic assembly is one of the original and most recognizable application of robotics, and there are countless examples of successes with the automation of factories of everything from cars to food [7, 8]. We view construction as an assembly process where the thing being built is static and the robots move. We present a selection of examples of platforms that could allow for this sort of approach.

The advent of online shopping has led to a invigoration of the mail-order model of sales, where a central set of warehouses deals with orders from customers over a much larger area than a store. This has led to a number of compelling examples of distributed and robotic warehousing systems which can store and recover items efficiently on demand [9, 10, 11]. Teller et. al. demonstrate a forklift which can semiautonomously manage a partially structured warehouse [12]. For much of this thesis, we treat the existence of some

depot which can provide parts as given.

The problem of mobile robots capable of manipulation and construction tasks is a very active area of research, and a great deal of robots have been demonstrated academically or are available commercially. Our work is implemented to run on any mobile manipulator with ROS drivers, which WillowGarage maintains a list of on their website [13]

2.2 Distributed Coverage

There is a significant body of related algorithmic work, especially relating to partitioning of spaces, [14] contains a good introduction to distributed partition of spaces under the section on coverage. In the construction space, a simple distributed 3D construction algorithm is described in [15], while [16] describes a 3D construction algorithm for modular blocks in a distributed setting. Stochastic algorithms for robotic construction with dependency on raw materials are analyzed in [17]. An algorithm for identifying and partitioning construction into subassemblies based on the relationship between parts is described in [18]. Previous work in on robotic construction from our lab includes Shady3D [19] utilizing a passive bar and an optimal algorithm for reconfiguration of a given truss structure to a target structure [1], and experiments in building truss structures [20].

Using Voronoi partitions to deploy robots for coverage was originally proposed in [21] and has been extended several times since then for tasks such as adaptive coverage [22] and equitable partitioning [23]. Our most recent work extends the idea of equitable partitioning and combines it with coordinated construction of truss structures [2], locational optimization [24], and adaptation to failure and shape change [25]. Recently in [26] Hsieh et. al. extended our work to consider the complexity of a target structure during construction to aid delivery, a topic we also address under a different set of constrains (our focus in this space is strong guarantees of completion of a task given multiple classes of constraint). We also use insights on the relationships between polygons in low-dimensional space gleaned from [27] and [28] in our partitioning algorithms.

2.3 Extentions on this work

The work outlined in this thesis, especially the constraint model outlined in chapter 3 was used as a basis for the work in [29, 30, 31].

Chapter 3

Problem Formulation

3.1 Overview: Distributed Robotic Construction

We consider the task of building some structure in a region Q using a heterogeneous set of robots divided into two classes: assembly robots and delivery robots. Delivery robots are robots devoted to retrieving parts and bringing them to the appropriate point in an assembly. Delivery robots take parts and attach them to some final assembly. Robots may be added or removed during runtime, either due to unexpected failures or some deliberate or external change in the availability of robots to the system. Our measure of success is the presentation of distributed algorithms which allow a structure to be built successfully while maximizing the amount of robots working at any given time. We should also be resistant to changes in the number of robots in the system.

We use a decentralized algorithm originally proposed by Yun. et. al. that coordinates the robot team to deliver parts so that the goal assembly can be completed with maximum parallelism [2]. Algorithm 1 shows the main flow of construction in a *centralized* view. In the first phase, assembly robots locate themselves 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 assembly robots. The assembly phase continues until there are no source component left or the assembly structure has been completed.

Algorithm 1 Construction Algorithm

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

3.2 Demanding mass

We assume that we have been provided with a blueprint that described the structure to be build. For any part v in the blueprint we define the *demanding mass* $\phi(v)$ to be the priority of that part. That is, a part with higher mass should be placed before parts with lower mass. In related and foundational work the function ϕ is smoothed to be continuous, typically by defining anything within the space occupied by a part as having constant positive mass and setting the ϕ of any empty space to zero.

3.3 Task Partitioning

Given multiple assembly robots, we would like to partition the space such that as many assembly robots are working at any time. To achieve this, we partition the incomplete tasks such that each assembly robot r is assigned a partition $\mathcal{P}_r$. We require that each partition is convex and non-overlapping to reduce the need for robots to solve multi-robot navigation problems. Partitions should also have equal mass to ensure that robots have similar workloads and queues, which requires real-time balancing to accommodate changes in the robots in the system. As with all algorithms, we only present algorithms that are fully distributed and require only single-hop communication.

Chapter 4

Preliminary Experiments

4.1 Foundational work

The system presented in [1] focused on truss structures built with two types of components: connectors and links in order to simplify exposition and figures. The work assumes that the robots move *freely* in an Euclidean space (2D and 3D).

Task Allocation by Coverage with Equal-mass Partitions

The problem formulation from [1] closely resembles the one discussed in chapter 3. Suppose n assembling robots cover region Q with the configuration $\{\mathrm{p}_1, ..., \mathrm{p}_n\}$, where p_i is the position vector of the i^{th} robot. Given a point q in Q, the nearest robot to q will execute the assembly task at q. Each robot is allocated the assembly task that includes its Voronoi partition V_i in Q.

$$V_i = \{\mathrm{q} \in Q | \left\|\mathrm{q} - \mathrm{p}_i\right\| \leq \left\|\mathrm{q} - \mathrm{p}_j\right\|, \forall j \neq i\} \tag{4.1}$$

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.

$$M_{V_i} = \int_{V_i} \phi_t(\mathrm{q}) d\mathrm{q} \tag{4.2}$$

Each robot follows its own local controller, designed to achieve a global distribution of robots so that each robot to have the same amount of assembly work. We call this *equal-mass partitioning*. The problem is challenging because the Voronoi tessellation evolves as robots act. Yun et. al. developed a decentralized controller for equal-mass partioning of spaces given a continuous, differential ϕ in [2]. This thesis presents a decentralized controller for equal-mass partitioning in Section 5.2

Delivery and Assembly Algorithms

Once the assembly robots are in place, construction may begin. During construction we distributed the source components (truss elements and connectors) to the assembly robots in a balanced way. Global balance, which is defined as balance of delivery to all the assembly robots, is asymptotically achieved by a probabilistic target selection of delivering robots that uses ϕ_t as a probability density function. For local balance defined for only neighboring robots, the delivering robots were driven by the gradient of demanding mass defined as the remaining structure to be assembled by the robot. Robots with more work to do get parts before robots with less work. Each assembly robot waited for a new truss element or connector and assembled it to the most demanding location in *its Voronoi region*. Therefore, construction is purely driven by the density function regardless of the amount of the source components. We ensured all control processes were distributed and robot communication was restricted to direct neighbors.

4.2 Platform

4.2.1 Experimental Testbed

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 (The blue half-circle plate in Figure 4-10). Trusses and connectors are manually supplied to the cache during

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

Table 4.1: Specifications of the robot

experiments. In 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.

4.2.2 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 4-1. Specifications of each component are in Table 4.1. The gripper of the arm has been replaced by an instrumented gripper which contains an infrared communication transceiver and is contoured to align a grasped part as the gripper closes. The special design allows the gripper to reliably grasp parts despite centimeter-scale uncertainty in a position of the parts, by passively aligning the grasp point into a unique 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. We equipped each robot with a small Dell Inspiron Mini 10s netbook which runs a Java-based controller.

4.2.3 Smart parts: Instrumented trusses and connectors

Smart parts enable grasping for robotic delivery and assembly via communication. We explore the use of communication as an alternative to using computer vision for part identification and grasping. IR communication devices are instrumented as shown in Figure 4-3 on the robots and within each parts. A part can guide a robot to its location and tell the robot its part type.

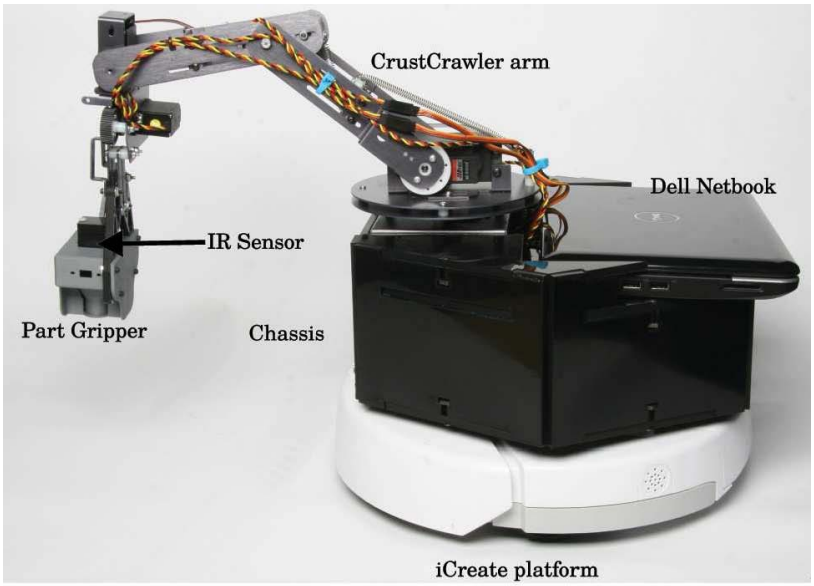


Figure 4-1: Side view of robot hardware with the Crustcrawler arm. From a fixed base, the arm allows for grasping an object on the ground in a half-arc in front of it with a depth of about 20cm.

Figure 4-2 shows two types of the smart parts: truss and connector. The connector is capable of connecting 6 trusses in the North, South, East, West, Up, and Down directions. Figure 4-4 shows a cube built from 8 connectors and 12 trusses. With a rechargeable 3.7v 210mAh lithium polymer battery, the parts weigh 60 grams. The truss is 18 cm long.

4.2.4 Infrastructure for localization and communication

For delivery and assembly, the robots receive precise location information from a Vicon motion capture system providing the 2D positions and the rotational heading with accuracy to the millimeter and milli-radian respectively at 10 Hz using a commercial xBee RF (radio frequency) wireless mesh network. Between the robots, a UDP multicast channel on the local network is implemented with a singe WLAN router. The UDP packets contain a logical time-stamp, a robot ID number, their current positions, and their current target robot. The robots also broadcast their states such as whether or not they are currently carrying or dropping off a part, which part type they are carrying, where they are carrying this payload, and the knowledge of any other known placed parts.

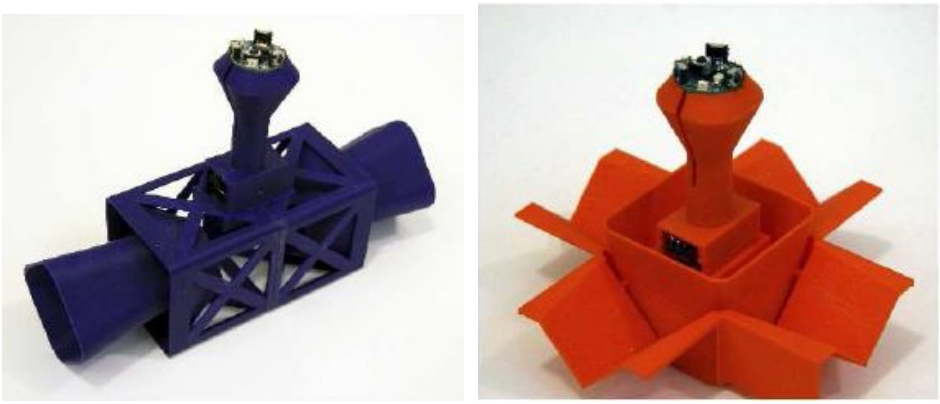


Figure 4-2: Smarts parts to be delivered: (LEFT) a red connector (RIGHT) a blue truss

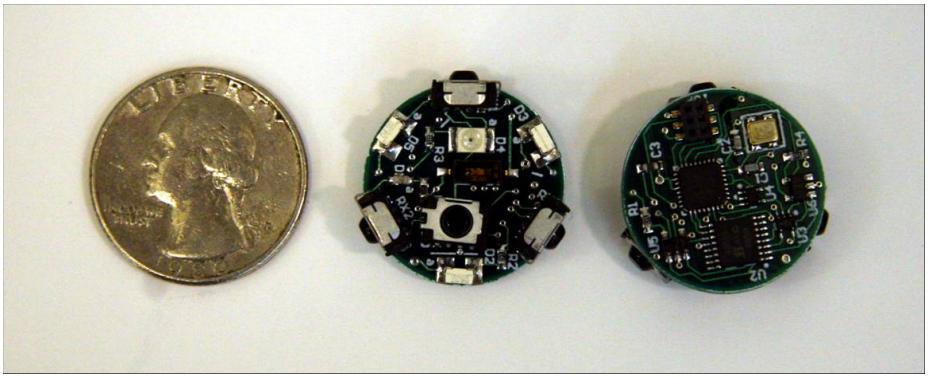


Figure 4-3: The small IR communication modules on a PCB that can be embedded in parts to create a smart environment for the robots to sense.

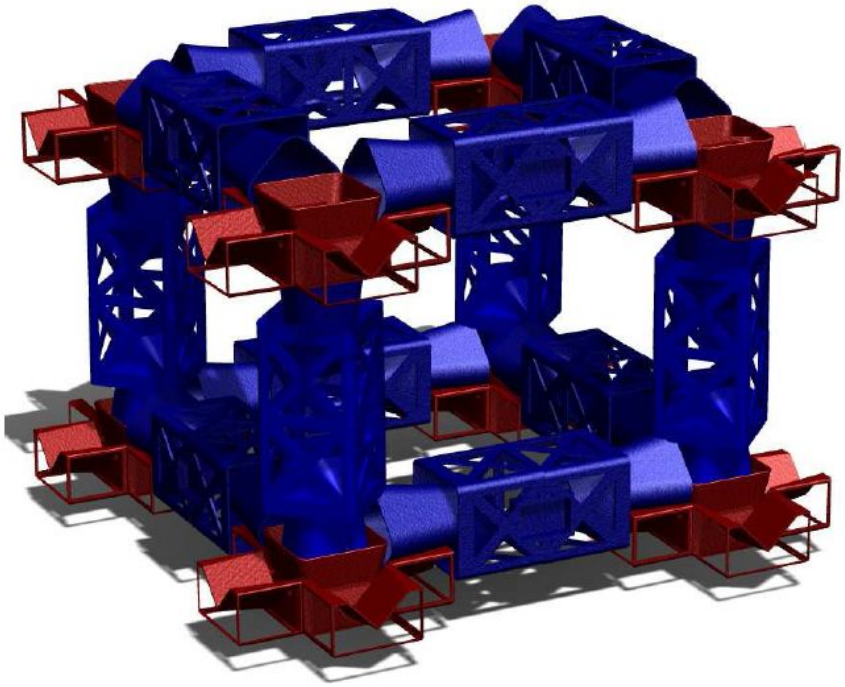


Figure 4-4: This 3D-rendered image of a cube is constructed from 8 junctions, and 12 struts.

4.2.5 Software architecture

The software architecture is structured hierarchically. The highest level planner can be swapped while using the same underlying modules. We use this modularity to create *assembly* and *delivery* planners, either of which can control the robot functions as shown in Figure 4-5.

Each software module is implemented in Java and runs in its own Java thread. The planner thread controls manipulation and motion of a robot. The planner gives the robot only an end destination and information on any obstacles, such as moving robots or parts on the ground. The planner waits for the motion to finish before trying to manipulate the arm, and gives the robot arm two commands: *pick up the part* or *put down the part*. The planner makes the decisions on where and when to move and manipulate parts by updating with the information received by the communication module. The communication module contains the most up to date information for the planner, which the planner uses to determine where

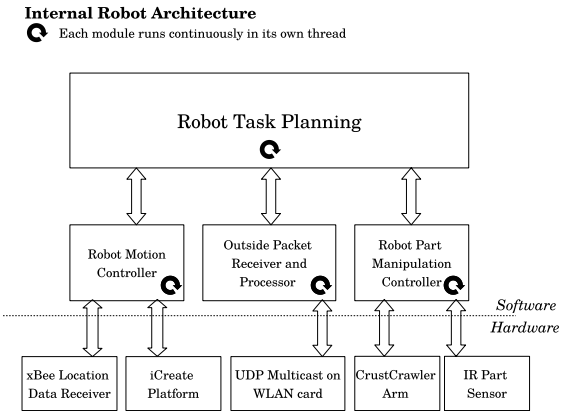


Figure 4-5: The hierarchical software architecture of the robot platform.

to move next. The planner is responsible for navigation, manipulation and communication modules commands , and these three modules handle low level control for the mobile, the arm, the manipulating IR sensors, and the communication messaging hardware.

4.3 From Theory to Practice

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 4.2. The most important components are manipulation and navigation, used both for assembly and delivery.

4.3.1 Delivery

The delivery algorithm, constructed as a finite state machine in Figure 4-6, follows the theory and takes steps to account for the real world challenges of multiple robot systems such as collision avoidance, asynchronous communication, and part dependencies. The robots have theoretical access to perfect information about the locations and demanding mass value of the surrounding robots, which we replace with a fault tolerant, asynchronous communication protocol to allow robots to learn about the surrounding parts and robots.

Experiment	Controller from [2]
• 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 acknowledgments 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 4.2: Controller from [2] Vs. Experiment

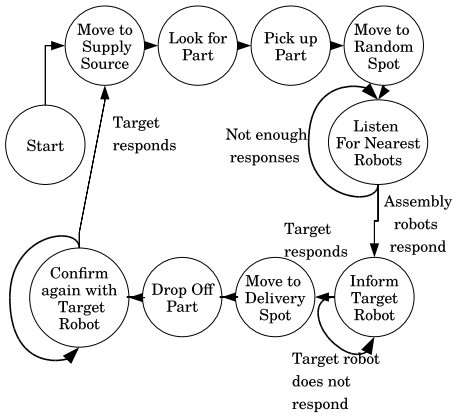


Figure 4-6: The task planning event loop for the delivery robots. The main loop pauses and loops back on itself at points where continuing requires asynchronous communication from other robots.

Algorithm 2 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.
```

Finally, the original algorithm assumes that the delivery order of parts will have no affect on the assembly of the structure. The practical delivery algorithm replaces the notion of parts as simple blocks with a model of parts as part of a blueprint, where the order in which parts are delivered can be factored into demanding mass calculated at any given time. These extensions to the algorithm allow it to be carried out on the physical system.

The system follows Algorithm 2 to complete its task, with the sub-modules taking over much of the error-handling. The navigation module, as discussed earlier, handles possible collisions while moving to the source and to robots, and it waits for the source to be clear of other robots before docking. The delivery robot acquires a specialized part from the supply source by performing a scan with its gripper. Asynchronous communication takes the place of actual gradient following when picking a location to deliver a part.

The model of the system as a blueprint of parts, chained together with dependencies, allows the assembly robots to look at the map, determine which parts are still needed at a given time, and request that number of parts to the delivery robots. This implementation does not change the delivery algorithm and helps prevent bottlenecks in spots where the demanding mass for the completed structure and the demanding mass at that moment are

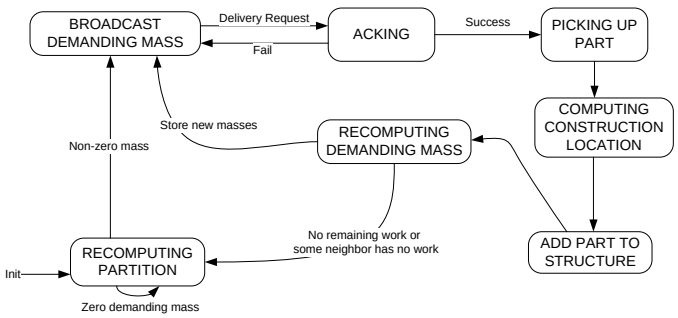


Figure 4-7: The task planning event loop for the assembly robots.

different.

4.3.2 Assembly

The assembly algorithm, demonstrated as a finite state machine in Figure 4-7, 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, 0and 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.

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 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.

4.4 Results

For testing platform, we use 2 assembly robots (labeled with robot 4 and 5) and 2 delivery robots (labeled with robot 2 and 3) in a 5x5 meter rectangle. The testing platform also involved a motion capture system to provide robot localization information and a GUI that gathers all the activities with communication and displayed them. Below we discuss the behavior of our robots over the course of these runs in terms of both our algorithm and practical considerations. Note that the system is decentralized except for the locational information from the Vicon motion capture system.

4.4.1 Test Scenario

For evaluation, a single blueprint is chosen demonstrating different features of a real system and the number and locations of the assembly robots change for different runs. We specialize the delivery robots further: one picks up truss parts only and one picks up con-

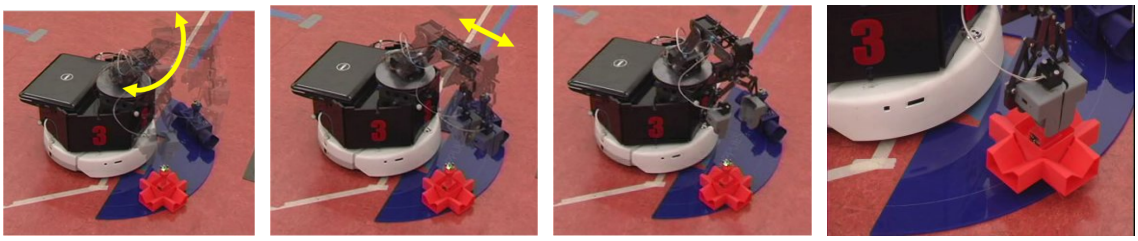


Figure 4-8: Snapshots of grasping. The arm moves along an arc to find a rough position of a part and does fine search by radial motion. Grasping is done after confirming the part.

nector parts only. The supply dock for parts is located at position (0,0), however the parts at the supply dock are moved around to test the robots' ability to pick up reachable parts. The robots could sense the different types of parts 100% of the time by communicating with them over IR.

4.4.2 Robot Adaptation

In an ideal setup and execution, the delivery robots alternate between the two assembly robots. To test adaptivity, we also run a variation in which one robot stops demanding parts halfway through the test. This failure of the assembly robots causes the delivery robots to adapt, delivering parts only to the remaining robot.

We ran the scenario with two assembly robots on the platform 12 times. All runs produced the correct alternating delivery behavior. Both the joint delivery robot and the truss delivery robot alternated targets and delivered to both assembly robots, seen in Figure 4-10. The delivery robots alternate targets in response to the demanding mass reported to them by the assembly robots, shown in Figure 4-9.

We ran the same scenario as before 3 times with a simulated failure, in which one of the assembly robots was taken off the map. Even when the assembly robot removed had a higher demanding for parts, its failure resulted in the delivery robots delivering to the remaining robot. In all cases, the communication between delivery and assembly robots confirmed the deliveries and changed the demanding masses of the assembly robots. Over

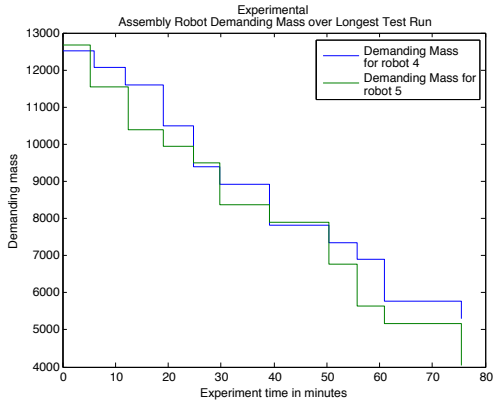


Figure 4-9: The demanding mass of assembly robots, named robots 4 and 5, drops whenever a part delivery occurs. Delivery robots changed targets to whichever robot had the highest demanding mass at the time. The unit of demaning mass is undimensional and proportional to amount of the partitions.

all 12 test scenario runs, the 2 delivery robots completed 45/48 delivery attempts. Three failed deliveries were the result of arm hardware failure on a single robot. A summary of test runs can be seen in Table 4.3.

4.4.3 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 4.3. 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 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.

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 4.3: Summary of Robot Delivery Test Runs

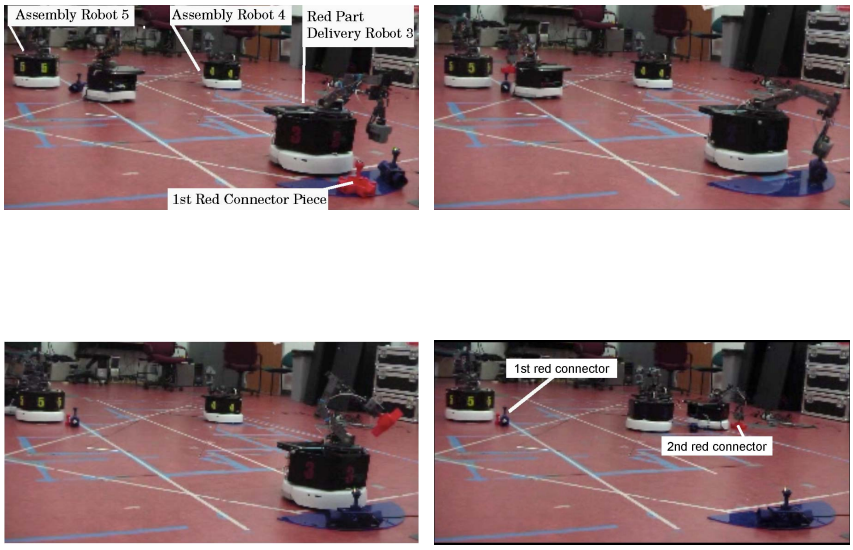


Figure 4-10: Snapshots of a test run of the even demanding mass delivery scenario. Assembly robots begin positioned at 2 different points of highest demand for parts. As the red connector parts are delivered, the maximum demanding mass for the entire map changes, causing the delivery robot to change delivery targets, first to robot 5, then to robot 4.

4.5 Identified Next Step

Translating this decentralized to a robotic platform revealed several areas where our assumptions from simulation needed to be reconsidered. The difficulties in converting from a theoretical system to a physical one fell into two major categories: assumptions about robots and assumptions about the assembly. The need for holonomic robots required new algorithms to be implemented to work around those assumptions. The lack of consideration of the physical constraints of the object being assembly mid-construction required us to place parts in patterns on the ground rather than actually build structures.

Chapter 5

Considering Physical Constraints

The experiments in chapter 4 exposed a need to build knowledge of the physical constraints on the mid-assembly stability of the target structure into our algorithm. We also noted the equal-mass partitioning requiring a continuous mass function despite the discontinuities inherent in assigning a mass to each individual part in an assembly. One of the researchers working on the experiments outlined in chapter 4 takes a first pass at these issues in [24] by adding a "reveal" function that hides demanding mass of parts and forces construction to occur in a predetermined order, and creating a "graph based Voronoi partition" scheme.

This chapter presents solutions to both of these problems by describing a discrete equal mass partitioning algorithm and a density function that allows parts to be prioritized in a fashion that guarantees convergence of the system and tends toward high parallelism.

5.1 Overview of Construction Algorithm

We are given a team of robots, a blueprint of a desired structure, and a construction region Q. A subset of n of the robots are specialized as assembly robots and the rest are specialized as delivery robots. The blueprint describes the location, type, and physical requirements for stability of each object ("part") in the structure. The robots are given a local section of a blueprint, and have full knowledge of the progress of the construction of the target structure in the area surrounding them and their neighbors. We describe an algorithm that coordinates

the construction of a given structure while maximizing parallelism across assembly robots and conforming to the physical constraints of the structure. The algorithm is guaranteed to construct the structure in an order that is feasible in that it does not prevent any sub-assembly from being completed.

This problem formation differs from previous work in coordinated construction discussed in chapter 3 and covered in ([25, 2]) in that we introduce knowledge of physical constraints on assembly and delivery. We consider two new constraints: the requirement that each part must be capable of staying in place once set (physical dependency), and the requirement that a delivery robot must be able to reach both the source and assembly robots at all times (reachability).

We represent these constraints in the blueprint by defining parts as vertices connected on two graphs: a physical dependency graph $G_p(V, E_p)$, and a reachability graph $G_r(V, E_r)$.

We define G_p by placing a directed edge $(v_i, v_j) \in E_p$ between any v_i and v_j where v_j cannot be stably placed unless v_i already has been, regardless of the state of any other part. We assume that any part is placeable iff all of its parents in G_p have been placed.

We define G_r by placing a directed edge $(v_i, v_j) \in E_p$ between any v_i and v_j where a robot at the location of v_j could access v_i, regardless of the state of other parts in the systems.

Without loss of generality and for ease of exposition, we assume that there is some constant upper bound c on the maximal degree of any vertex in V.

Algorithm 3 outlines the construction algorithm from a global perspective, and is a reproduction of the foundational algorithm (algorithm 1) presented in chapter 3 with the partitioning step omitted, as task allocation is now computed in real time (algorithm 3 line 4 and section 5.2). The three primary functions performed by the robots are partitioning, delivery, and assembly. Assembly robots are deployed into Q, and the partitioning controller (Section 5.2) ensures that each assembly robot i has some partition of $\mathcal{P}_i \subset V$ that is spatially compact and that each partition has roughly the same amount of work assigned to it at all times. Delivery robots constantly deliver parts from some source to the assembly robots, using a priority function to determine the best place to deliver each part within a

local region (Section 5.3).

Algorithm 3 Construction Algorithm

1: Deploy assembly robots in Q
2: **repeat**
3: **delivering robots:** deliver source components to assembling robots
4: **assembling robots:** assemble the delivered components while constantly updating partitions
5: **until** task completed **or** out of parts

5.2 Equal Mass Partitioning using a Discrete Approach

In our problem formulation we represent each part in the target structure as a point, which is reasonable given the discrete nature of parts. We define the *demanding mass* of a part as a measure of its priority in placement order, where the mass of a part is 0 if a part is unplaceable or already placed and positive otherwise (this is discussed further in section 5.3). By partitioning based on this mass function, we can allocate roughly the same amount of reachable, actionable work to each robot. We repeat this algorithm continuously during runtime to maintain an equitable partitioning of Q as masses change dynamically while the structure is built.

A trade-off of the significant increase in fidelity we get by updating our model from a geometry to a blueprint is a change in the nature of the density of the Q. The density of Q is used by most coverage algorithms, including canonical Lloyd algorithms for equipartitioning, to perform gradient descent to converge to equal-mass partitions. Our blueprint forces the density of Q to be a dynamic summation of scaled Dirac delta functions, which has a gradient of either zero or infinity at all points, meaning we can not use the class of deployment algorithms that depend on Voronoi partitioning.

The problem of finding a equal mass convex partitioning of a set of points is NP-complete (proof provided as Appendix A). Here we present an algorithm that efficiently finds a local minimum of convex partitions. That is, in a stable configuration found by this algorithm no point in the convex hull of a partition can be placed in another partition

without either requiring a non-convex partition to exist or increasing the disparity between two hulls

5.2.1 Algorithm

Vertex swap, which we present a potential solution to this problem in [24], works on a graph rather than in $\mathbb{R}^n$, and requires multi-hop communication. In order to use this algorithm, we need to define a graph that connects the set of positive mass points. If we create a relatively sparse graph we introduce unnecessary assumptions which limit which points can be in the same partition. If we create a well-connected graph we introduce the assumption of excessively large communication radii as neighbors are defined by edges in the graph rather than L^p distance. We have developed a equipartitioning algorithm that does not require a graph connecting points, uses only local communication, and has lower complexity than vertex swap.

We identify partitions that are spatially compact and approximately equal mass, but as stated above Voronoi partitioning and vertex swap are not viable options. The problem of partitioning a set of point masses in $\mathbb{R}^n$ into non-intersecting, convex, equal-mass partitions is NP-hard, even in $\mathbb{R}^2$ (Appendix A). We present the *hull vertex swap* algorithm (Algorithm 4), an efficient distributed method for approximating equal-mass partitioning using only single hop communication.

Hull vertex swap converges to a convex partitioning of the points $v \in V$ distributed across the space Q into a set of partitions. We allow each partition $\mathcal{P}_i, i \in [1, n]$ to "steal" points from its set of neighbors $\mathcal{N}_{\mathcal{P}_i}$. The focus of the algorithm is to determine which vertices can be transferred from one partition to another without creating an intersection between the convex partitions, and which vertices can be stolen to effectively converge to a solution that locally maximizes our measure of equality.

We now discuss how to determine which vertices can be stolen without introducing intersections between partitions. We then present how to compute which vertex is best to steal, if any, and finally present a proof of convergence and experimental data. To compute which vertex to steal, each robot first computes the convex hull of its partition $\mathcal{P}$; then for

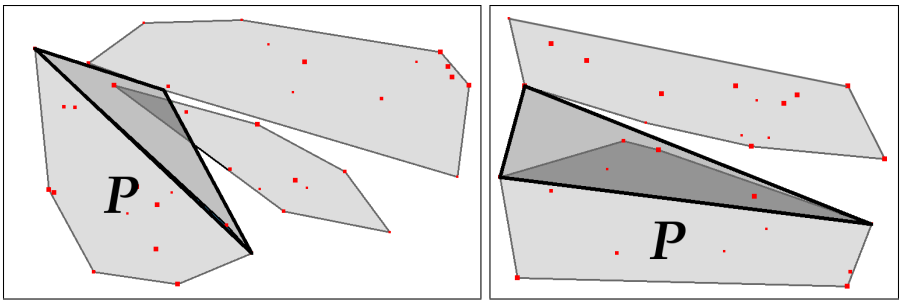


Figure 5-1: Test to identify stealable vertices. The triangles with bold outlines mark the region that would be added to $\mathcal{P}$ due to a trade of a vertex. (left) Adding the vertex would cause a collision between two polygons. (right) This vertex would be considered a valid candidate to trade.

each vertex v_i in the hulls of its neighbors, it considers the region that would be added to the polygon defined by the convex hull of $\mathcal{P}$ if v_i were moved into $\mathcal{P}$. Any v_i that would not create an intersection between two polygons if added to $\mathcal{P}$ is considered a *stealable* vertex. The area added to the region can be quickly tested for intersection by finding the triangle formed by the tangent rays between $\mathcal{P}$ and v_i and testing the edges of each of the hulls in $\mathcal{N}_\mathcal{P}$ for intersection with that triangle (see Figure 5-1). In higher dimensional cases this extends to the pyramid formed by tangent planes.

Algorithm 4 Partitioning Algorithm

Deploy into Q at random pose p_i
$\mathcal{P} \leftarrow \{v|(||pose(v) - \mathrm{p}_i|| < ||pose(v) - \mathrm{p}_j||)\forall j \neq i\}$
loop
  compute convex hull of $\mathcal{P}$
  update $\mathcal{N}_P$
  $X \leftarrow \{v|v \in \mathcal{N}_\mathcal{P}, v \text{ is stealable}\}$
  $i \leftarrow \underset{v_i \in X}{\operatorname{argmax}}(\Delta\mathcal{H}_\mathcal{P}(v_i))$
  if $\Delta\mathcal{H}_\mathcal{P}(v_i) > 0$ **then**
    communicate to $\mathcal{N}_j : v_i \in \mathcal{P}_{\mathcal{N}_j}$ to remove v_i
    $\mathcal{P} \leftarrow \mathcal{P} \cup v_i$
  end if
end loop

We measure equality using a cost function $\mathcal{H}$ from [21] with a constant distance func-

tion. Given that each vertex v has a mass $\phi(v)$:

$$M_{\mathcal{P}} \triangleq \sum_{v \in \mathcal{P}} \phi(v) \tag{5.1}$$

$$\mathcal{H}_Q = \prod_{i \in [1,n]} M_{\mathcal{P}_i} \tag{5.2}$$

Without loss of generality, if we consider moving a vertex v from $\mathcal{P}_1$ to $\mathcal{P}_2$, we can compute the change in mass:

$$\Delta\mathcal{H}_Q = \left(\prod_{i=3}^{n} M_{\mathcal{P}_i} \right) \left(M_{\mathcal{P}_2} + \phi(v)\right) \left(M_{\mathcal{P}_1} - \phi(v)\right) - \mathcal{H}_Q \tag{5.3}$$

When comparing two potential exchanges of vertices, we only need knowledge of the partitions that will change in order to compute both the sign and relative magnitude of our deltas. We therefore need only local knowledge to determine which vertex, if any, is best to trade. We can therefore compute a scaled local $\Delta\mathcal{H}_{\mathcal{N}}$ of moving some v from some neighbor's partition $\mathcal{P}_i$ to $\mathcal{P}_{self}$ with:

$$\Delta\mathcal{H}_{\mathcal{N}} = \left(\prod_{\mathcal{P}_k \in \mathcal{N} \wedge \mathcal{P}_k \neq \mathcal{P}_i} M_{\mathcal{P}_k} \right) \left(\phi(v)(M_{\mathcal{P}_i} - M_{\mathcal{P}_{self}} - \phi(v))\right) \tag{5.4}$$

$$\Delta\mathcal{H}_{\mathcal{N}} = \frac{\Delta\mathcal{H}_Q}{\prod_{\mathcal{P} \notin \mathcal{N}_{\mathcal{P}_{self}}} M_{\mathcal{P}}} \tag{5.5}$$

5.2.2 Analysis and Experiments

Convergence

Theorem 1 *Algorithm 4 will converge to a local maximum.*

Proof 1 *We know that the denominator in equation 5.5 will be unchanged by a vertex being*

stolen and that therefore

$$\underset{v_i \in X}{\operatorname{argmax}}(\Delta\mathcal{H}_{\mathcal{N}}(\mathcal{P} \leftarrow v_i)) = \underset{v_i \in X}{\operatorname{argmax}}(\Delta\mathcal{H}_{Q}(\mathcal{P} \leftarrow v_i)) \tag{5.6}$$

so each stolen vertex will result in an increase in $\mathcal{H}_Q$. The value of $\mathcal{H}$ is bounded from above and all $|\Delta\mathcal{H}|$ is bounded from below, so by induction the algorithm must converge to a local maximum.

Runtime

Theorem 2 *Update at each step of algorithm 4 runs in $\mathcal{O}(||\mathcal{N}||^d + ||\mathcal{N}||||\mathcal{P}||)$ time.*

Proof 2 *Consider a single step of algorithm 4 running on a robot in $\mathbb{R}^d$. Finding a triangle or cone takes $\mathcal{O}(||\mathcal{P}||)$ time. Checking for intersections takes $\mathcal{O}(||\mathcal{N}||^{d-1})$. This check needs to be run on $\mathcal{O}(||\mathcal{N}||)$ candidate points [28]. Computation of each $\Delta\mathcal{H}$ takes constant time, so the computation of candidate points dominates this function. The runtime per step is therefore $\mathcal{O}(||\mathcal{N}||(||\mathcal{N}||^{d-1} + ||\mathcal{P}||)) = \mathcal{O}(||\mathcal{N}||^d + ||\mathcal{N}||||\mathcal{P}||)$.*

Because only the hull is considered, this is often much faster in practice.

Experiments

We ran the partition algorithm on several hundred randomly generated sets of pointmasses with random mass. Point location was sampled from either a uniform distribution or 2D Gaussian. The partition masses converged on all pointsets such that their standard deviation was less than twice the average mass of a point. No partitionings contained outliers after convergence, which suggests that most local maxima are good approximations of equal-mass partitioning (see Figures 5-2 and 5-3). The simulations took 15.5 minutes in an environment with 500 point masses with 12 robot state machines each running in a separate thread on a single 1.2 GhZ core. Running the same environment with 5 robots converged in 2.5 minutes, and with 5 robots and 250 points the system converged consistently in under 45 seconds.

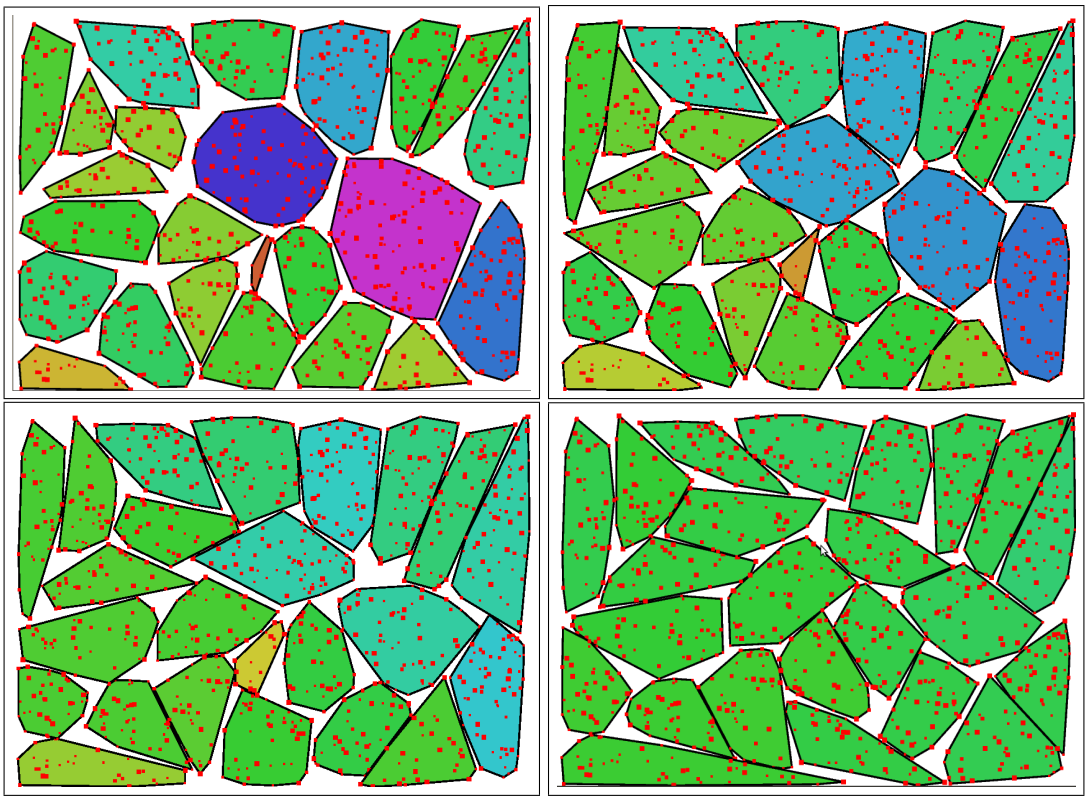


Figure 5-2: Data from running partitioning algorithm. Initial configuration (top left), configuration after 6 time-steps (top right), after 6 time-steps (bottom left), and after 26 time-steps (bottom right) on a set of point-masses with random location and mass. Shade is a function of total mass of a partition.

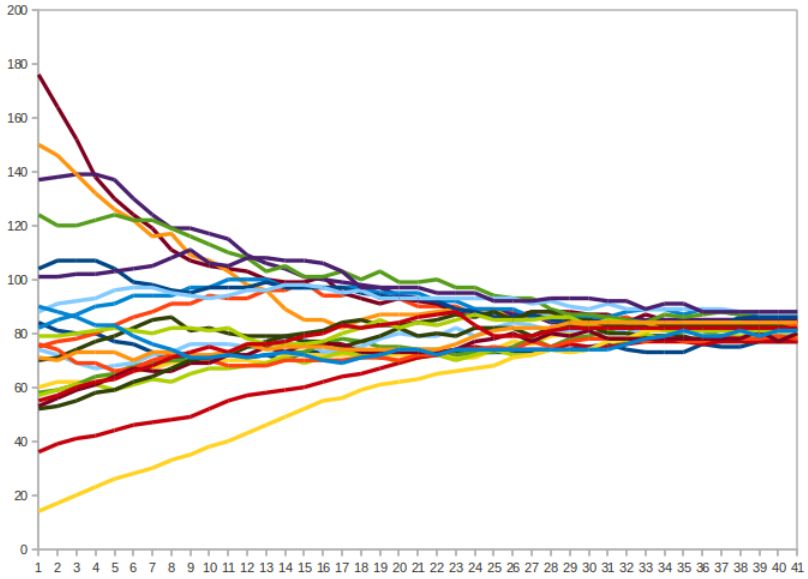


Figure 5-3: Total mass of each partition over time during a random run of the partitioning simulator.

5.3 Delivery and Assembly with Part Ordering

Delivery robots repeatedly choose random assembly robots and deliver the part with the highest demanding mass inside the chosen assembly robot's partition. The assembly robot waits for a delivery and then performs whatever actions are necessary to attach the part to the main structure.

Algorithm 5 Delivery Algorithm

1: **loop**
2: Move within communication range of random assembly robot r
3: Receive highest priority vertex in $\mathcal{P}_r$ from r
4: Bring corresponding part from part source to r
5: **end loop**

5.3.1 Algorithm

In our definition, parts with 0 mass violate either physical or reachability constraints. Between any two parts with non-zero mass, the part with higher mass is given priority in

Algorithm 6 Assembly Algorithm

```
Start partition algorithm (Alg. 4)
loop
    for v ∈ P_self do
        if v reachable from outside construction site then
            dist(v) ← 1
        else
            dist(v) ← 1 + min({dist(u)|(u, v) ∈ E_r})
        end if
    end for
    yield until delivery
    receive delivery of part v
    place v and signal neighbors
    for u ∈ all children and parents of v do
        update φ(u) (Equation 5.23)
        for w ∈ all children and parents of u do
            update φ(w)
        end for
    end for
end loop
```

placement. Given this planning algorithm, the mass function $\phi(\cdot)$ dictates the order in which parts are placed. We need a mass function with the following properties:

- no part placement violates global constraints
- after a part is placed the number of placeable parts tends to increase or remain constant
- the creation of bottlenecks and hallways is avoided if possible
- changes to the local density function can be efficiently calculated and updated using only local information

The precise order in which parts are placed is partially a function of the assignment of partitions and availability of parts, which are respectively non-deterministic and outside of our control. The ordering should optimize over some set of local metrics. To build this function, we present mass functions that each satisfy one of our goals and then describe a

combined definition. In each definition we represent the placement of a part by removing the vertex v_i corresponding to the part placed and also removing any edge going into or out of v_i from both graphs.

Before defining our mass function we need to make a modification to the reachability graph. We need local information about the global property of reachability, and one way to do this is to modify reachability into a DAG. We do this by defining $G'_r(V, E'_r)$ such that:

$$E'_r \triangleq \{(u,v) | (u,v) \in E_r \wedge dist(u) > dist(v)\} \tag{5.7}$$

We are now ready to begin defining the mass function ϕ. First we define the global constraints formally: any v_i is placeable iff it will be physically supported and not render any unplaced parts unreachable. We define two boolean variables $\xi_p(v)$ and $\xi_r(v)$ to represent this criteria.

$$\xi_p(v) = (deg^-_{G_p}(v) \neq 0) \tag{5.8}$$

$$\xi_r(v) = (\exists j : ((v_j, v) \in E'_r) \ \wedge \ (deg^+_{G'_r}(v_j) = 1)) \tag{5.9}$$

ξ_p indicates that a part will not be physically supported if its indegree is anything but 0; all the parts it depends on for support must already be placed. ξ_r indicates that the part should not be placed if doing so would prevent delivery robots from reaching another part; that is, if placing a part blocks a unique exit it cannot be placed.

$$\phi_c(v) = \begin{cases} 0 & \xi_p(v) \vee \xi_r(v) \\ 1 & otherwise \end{cases} \tag{5.10}$$

Because the ordering of parts is defined by a set of DAGs, any mass function that obeys the constraints above and sets all other $\phi(v_i)$ to a positive value will terminate if the problem is solvable. This is sufficient to have a system that will build a structure without violating any physical constraints, however with binary mass placement order will be essentially random.

The remaining mass functions allow behavior to be tuned to tend towards placement that allows for better parallelism of assembly tasks and access to the structure by delivery robots.

Before presenting these functions, we introduce the following scoring function and briefly discuss its properties. Given some function $f : x \to \mathbb{Z}^+$, and some candidate sets X_i with the property $||X_i|| \leq c_X \forall i$:

$$score(f(\cdot), X) = \sum_{x \in X} \left(2^{f(x)}\right)^{-c_X} \tag{5.11}$$

This function takes advantage of the properties of geometric series to provide a function that will, given two candidate sets X_1 and X_2, give a higher score to the set with the most values generating the lowest value of f. That is, if we identify the lowest value of $f(x)$, y which is generated by a different number of members of the two sets:

$$y = \min_i \{i : ||\{x \in X_1 | f(x) = i\}|| \neq ||\{x \in X_2 | f(x) = i\}||\} \tag{5.12}$$

then $score$ has the property that

$$score(f, X_i) > score(f, X_j) \tag{5.13}$$

implies

$$||\{x \in X_i : f(x) = y\}|| > ||\{x \in X_j : f(x) = y\}|| \tag{5.14}$$

For example: consider two nodes on a directed graph with sets of children X_1 and X_2, and a function $f(v)$ which returns the outdegree of a node. The node with more children that have outdegree 0 will have a higher score ($score(f, X_i)$). In the case of a tie, the node with more children with outdegree 1 will have a higher score. After that ties are broken by the number of children with outdegree 2, and so on. We use this function extensively in our definitions.

First we define a function that will help to place parts such that we first maximize the number of parts still available to be placed (i.e., reveal as many new parts as possible). A reasonable function could rank parts first by the number of physical dependencies they satisfy. We can represent this ranking with the $score$ function:

$$\phi_p(v_i) = score(deg^-_{G_p}, \{v_j | (v_i, v_j) \in E_p\}) \tag{5.15}$$

Similarly, we would like to place blocks that are least likely to cause a bottleneck first. By rating blocks by the number of different ways to reach their children we can place preference against restricting high-traffic paths. We also would like to tend toward placing parts in harder-to-reach locations first, so we need to define a slightly more complex test function $g(v_i) = \max_j(deg^+_{G'_r}(v_j)) - deg^+_{G'_r}(v_i)$.

$$\phi_r(v_i) \sim score(g, \{v_j | (v_j, v_i) \in E'_r\}) \tag{5.16}$$

We also would like to tend toward working in areas far from the easily reachable edge of the system first (i.e., at the end of a hallway). We can use the distance function from Algorithm 6 to measure this:

$$\phi_r(v_i) \sim dist(v_i) \tag{5.17}$$

To combine these two statements we normalize the distance function to between $\frac{1}{2}$ and 1. The score function behaves such that multiplying by a half is the equivalent of redefining the input function $f'(\cdot) = f(\cdot) + 1$. In this case doing so would effectively lower the outdegree of each of a node's children by 1, thus lowering the node's priority. This allows us to scale ϕ by distance without breaking the tiered behavior of the score function.

$$k_{dist}(v_i) = \frac{dist(v_i) - 1}{2(\max(dist(v) \forall v \in V, 2) - 1)} \tag{5.18}$$

$$\phi_r(v_i) = k_{dist}(v_i) score(g, \{v_j | (v_j, v_i) \in E'_r\}) \tag{5.19}$$

Finally, in combining these three measures of mass, we need to rescale our masses to allow comparison between ϕ_r and ϕ_p. To achieve this we introduce two scaling factors: β which rescales the range of in-degrees of nodes in E'_r to match that of E_p, and γ which can prioritize reachability or physical dependency as required by the task. The exact tuning of these functions varies depending on the capability and number of each class of robot, and this relationship is left as future work.

$$\beta = \frac{\max\limits_{v_i}(deg^-_{G_p}(v_i))}{\max\limits_{v_i}(deg^+_{G'_r}(v_i))} \tag{5.20}$$

$$\gamma \in \{-1, 0, 1\} \tag{5.21}$$

If we define $g'(v_j) = \beta(g(v_j) + \gamma)$, we can introduce those scaling factors to the reachability function by substituting into equation 5.19, which will normalize it to resemble the physical dependency function:

$$\phi'_r(v_i) = k_{dist}(v_i) score(g', \{v_j | (v_j, v_i) \in E'_r\}) \tag{5.22}$$

We can now combine equations (5.22), (5.15), and (5.10) to define our combined mass function for use by the controller.

$$\phi(v_i) = \phi_c(v_i)(\phi'_r(v_i) + \phi_p(v_i)) \tag{5.23}$$

5.3.2 Runtime

Upon the placement of a part, at most c parts will have a change of degree, which in turn means only c^2 parts have a potential change in mass. This allows constant time for a robot to update all masses after a part has been placed.

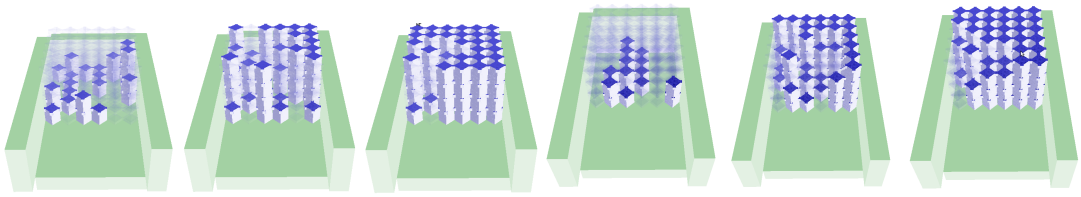


Figure 5-4: Part placement while building a solid cube using (top) uniform mass and (bottom) ordering. Note that without the ordering algorithm, work in the front occurs first (top middle), making it harder for delivery robots to reach subassemblies in the back. Also note how more of the stacks of blocks in the top right have reached their maximum height, leaving less opportunities for parallelism. A more detailed discussion of a simulator trail is given in Section 5.4.1.

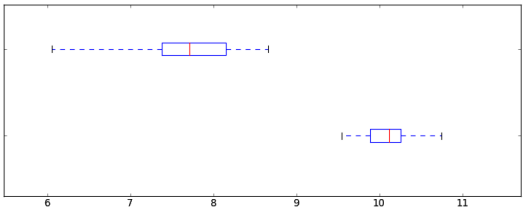


Figure 5-5: The average number of parts with positive mass across time over 50 runs of building a solid cube at the end of a hallway with 5 assembly and 4 delivery robots. (top) With uniform mass on placeable parts and (bottom) using the proposed algorithm.

5.3.3 Convergence

Theorem 3 *The controller outlined in algorithms 5 and 6 will converge to a complete structure if possible.*

Proof 3 *Our constraints are described by two DAGs. The mass function we describe here gives positive mass to all vertices with no unplaced parents, which by definition describes and follows a valid topological ordering of both G_p and G'_r, and will therefore converge without violating either sets of constraints.*

5.4 Experiments in Simulation

We have implemented Algorithm 5 and 6 in simulation and evaluated them on several construction test cases. We used two structures: one which demonstrates the properties of the controller in a high-stress scenario, and one which demonstrates a structure that might be encountered in actual applications. Each environment was tested at least 50 times for each possible permutation of between 1 and 5 delivery and assembly robots on three environments - an empty box and solid box at the end of a hallway (Figure 5-4: complex reachability, relatively simple physical dependencies) and a model airplane (Figure 5-6: complex physical dependencies, less complex reachability). All runs completed construction without violating constraints. We rated each run on availability - the number of parts ready to be placed - and throughput - the maximal number of delivery robots that could make a simultaneous delivery.

As a baseline, we compared the results of the mass function (equation 5.23) with the minimal constraint (equation 5.10). We saw a clear performance advantage to our algorithm in creating opportunities for parallelism (Figure 5-5). The throughput in our example situations did not exhibit statistically significant deviation from the uniform mass functions until late in each run.

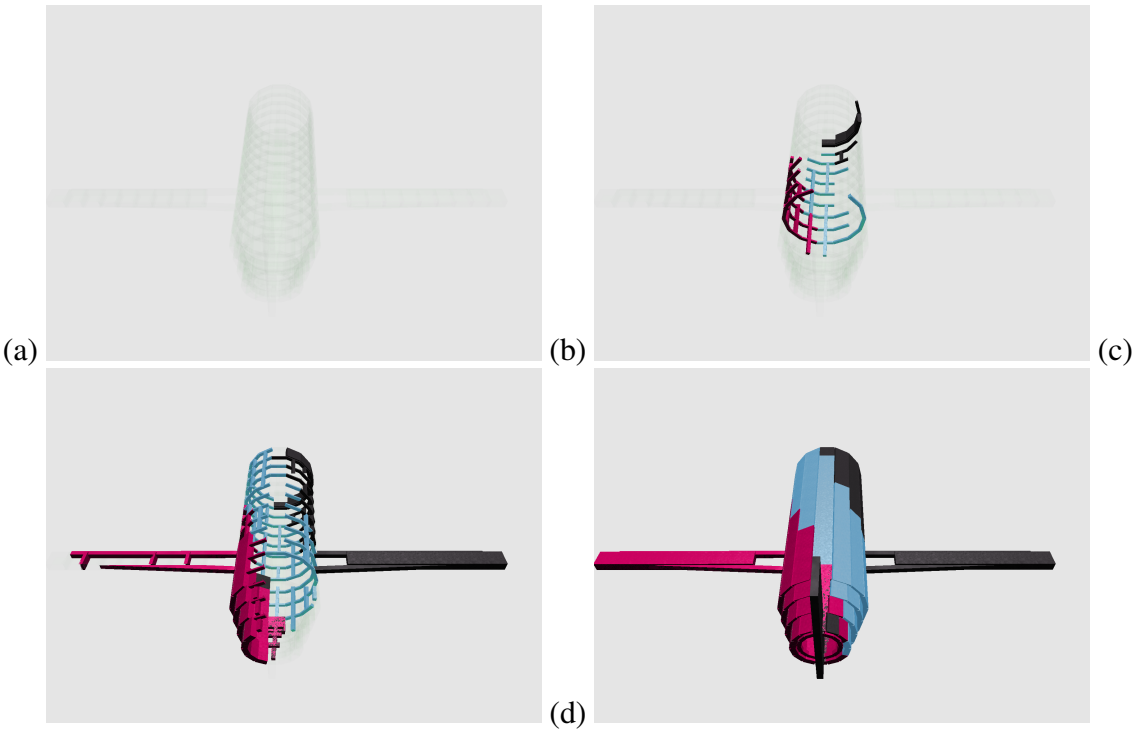


Figure 5-6: The controller running on three assembly and two delivery robots building a model plane in simulation, which has complex physical dependencies and relatively simple reachability constraints. Parts are color-coded by the robot that placed them.

5.4.1 Simulation Example: Model Plane

As an example of the behavior of the system we considered building a model plane with complex physical dependencies and a solid block which has complex reachability constraints. For illustration, we have colored parts differently if they were placed by different assembly robots. In this example, there are several constraints:

- The scaffolding must be completed before panels are placed over it
- Scaffolding must be built ground up, and have horizontal struts holding it up before it is more than 3 struts tall
- Wings require stable scaffolding before they are built, and the attachment point of main wing supports is unreachable after panels have been placed over the scaffolding around the connection point.

We initialize the system with three delivery and two assembly robots. The plane blueprint consists of 686 parts and 2402 edges which fully represent the dependencies described above. The dependencies were precomputed by brute force, which took slightly over one second on a 2.67 GHz processor.

In the screenshots from the simulator in Figure 5-6, these constraints are followed, and each assembly robot does roughly the same amount of work. In (a) we see an empty blueprint as the assembly robots are deployed. In (b) construction begins, with each of the three partitions filling up with roughly the same amount. In (c) the scaffolding has been mostly completed before panels are added, which maximizes the amount of potential work to be done and would allow for efficient parallelism if more robots were added. (d) shows the finished product, a completed airplane. The three robots placed 220, 232, and 234 parts respectively over the course of this example. It is also notable that when the robot placing dark grey parts completed its section of the assembly task it relocated to the still incomplete tail and resumed construction there, which was a direct result of the active partitioning algorithm.

Chapter 6

Feasability Experiments

We reimplemented algorithm 1 using the subroutines described in chapter 5. We consider a structure with a single class of parts, present a platform-independent implementation of our algorithm that will run on any robots that can be run by properly configures ROS nodes, and demonstrate it on an updated version of the platform from chapter 4. In **??** we demonstrate the generality of these algorithms by repeating the experiments on KUKA-brand youbots with more complex structures.

6.1 Software

Our software is implemented on top of the ROS infrastructure [32]. ROS (Robot Operating System) is a communication layer that allows the development of "rosnodes" which can run and communicate with each other on a single machine or between robots. We use this system to define rosnodes which to handle the different partitioning, ordering, and high level behavior tasks using generic localization data and then pass it off to any robot capable of manipulating parts appropriately (see figure 6-4). For ease of implementation, we changed some implementation details when implementing our system, as outlined in 6.1.

For working with our physical platform, we implemented robot drivers for the iRobot Create for motion and lynx motion crust-crawler arms for grasping. We use the Vicon

Figure 6-1: The “box” and “wall” shapes built in our experiments. “box” tests the ordering based onphysical dependancies, “wall” tests maintanance of reachability.

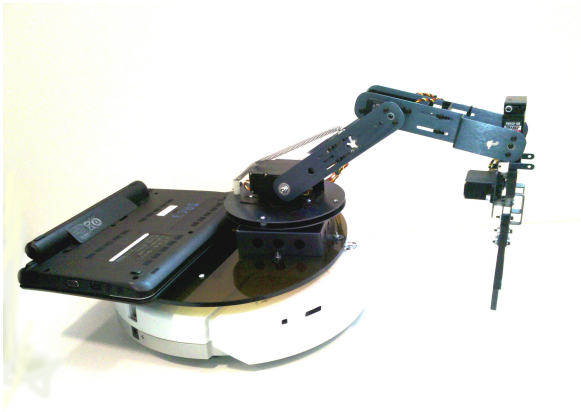


Figure 6-2: The new platform uses a different gripper and no longer has the augmented chassis, but uses the same technologies as the previous iteration.

Aspect	Theory	Implementation
Delivery	Gradient descent to local assembly robot with highest demanding mass	Delivery to robot with highest demanding mass
Assembly	Abstract assembly task	Concrete assembly task: part placement
Partitioning	Discrete partitioning with hull vertex swap	Discrete partitioning with hull vertex swap
Ordering	Part priority calculated in real-time based on dependencies, reachability, part supply, and assembly time	Part priority calculated offline based on dependencies, reachability, and part supply
Communication	Neighbors only (single hop)	All robots

Table 6.1: Summary of differences between our theoretical algorithms and system implementation.

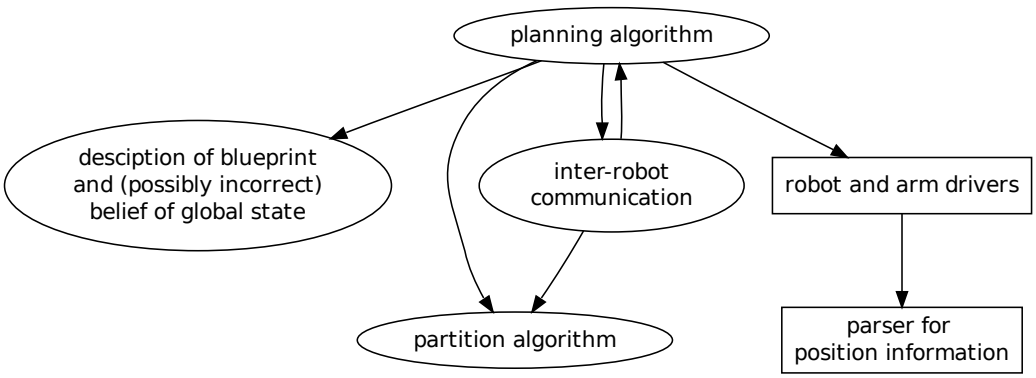


Figure 6-3: Dependency chart of software. Each node represents a ROS node, each arrow shows a dependency from parent to child. Square nodes must be implemented specifically for an environment, oval nodes are generic.

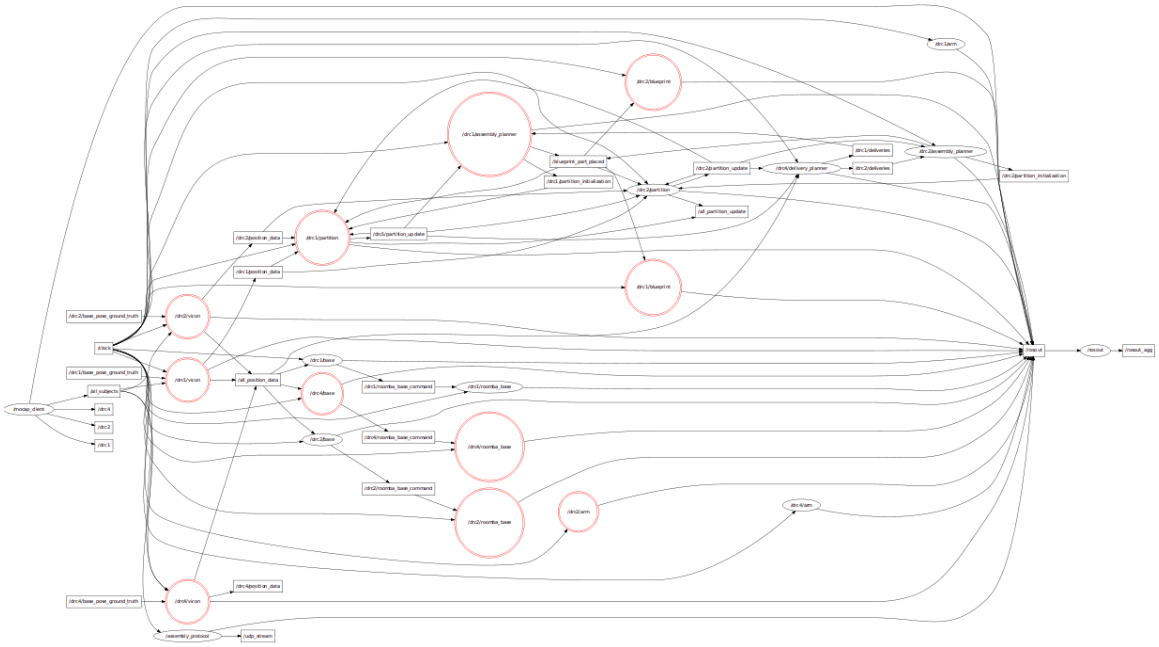


Figure 6-4: Auto-generated graph of information flow during a three robot experiment between nodes. Running our communication layer on a single machine increases the complexity of information flow, but allows for a central log of a distributed system

localization system with a UDP broadcast which we catch on each robot with a ROS node wrapper. Communication is handled by an independent central machine, which can allows us to monitor all communication and commands for logging.

6.2 Platform

Having observed the large delays at the source in the previous iteration of experiments, we chose to treat the problem of retrieving parts from a source as solved to allow a stronger focus on the task of construction. This removed the need for the smart part technology for these experiments, so our parts are significantly lighter and have no need for batteries. The removal of smart parts introduces the need to rely on the localization system to communicate where the now "invisible" parts are located. This reduced the reliability of grasping tasks, and so we allowed intermittent human intervention to nudge parts. In order to increase accuracy in the construction of the projects, the floor and parts are magnetized,

Figure 6-5: Our experimental platform while building a box.

which provides slightly over two centimeters of compliance for part placement. We also made several changes to the form of the chassis to ease access to the internals of the robots for maintenance.

6.3 Results

Most of the differences between theoretical robots and actual robots noted in chapter 4 are present in a system that notes part ordering, and so those observations and solutions largely remained unchanged for these experiments. Our assumptions about the structure itself did not require any fixes to translate into physical or simulated execution, which was a promising validation of the effectiveness of the part ordering scheme. Because we dealt with small sets of parts and robots in these experiments the partitioning algorithm did not show much change, as the part ordering system rarely showed more than two high priority parts in any shape, which led to an obvious correct partitioning, which our algorithm converged to in a single iteration of the swap action (presented in section5.2).

Though the changes to the platform improved runtime (average delivery time fell under 3 minutes), our results did not provide any new knowledge not presented in chapter 4.

The compelling results of our experiments were the 8 successful constructions of the box shape and 4 successful construction of the wall shape. This work provided validation of our approaches ability to autonomously conform to stability and reachability constraints.

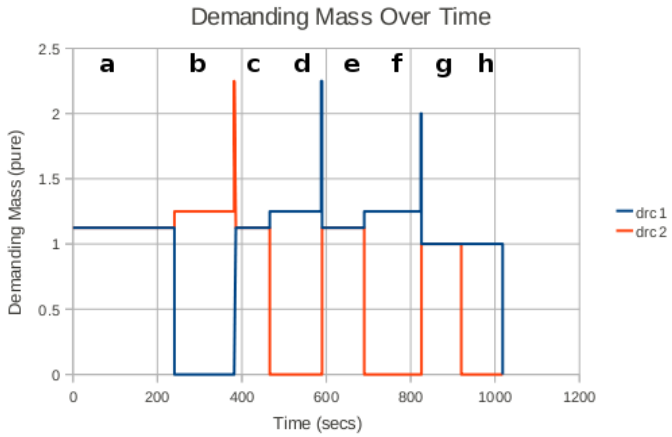


Figure 6-6: The demanding mass of each robot over the construction of a box. (Labels correspond to states symmetrical to those in figure 6-8, though they may be rotated by 180 degrees)

6.3.1 Walkthough of single iteration

For sake of illustration we will review the system state over an instance of the system constructing the "box" object. Figure 6-8 shows the state of the system over time. The assembly robots constantly repartition the parts that can be placed, based on their demanding mass. The total demanding mass for each robot is plotted in figure 6-6. We can see that after the first part is placed (figure 6-8.b), the demanding mass of one robot goes to 0 (as there is only one placeable part) and the other increases slightly (as the remaining part will make more pieces available, which increases its mass). After the next placement an assembly robot's total mass spikes as two parts are added to its partition, and then quickly levels out as the assembly robots equalize their partitions by swapping a part. This repeats at each level of the "box" structure. On the final layer, the mass does not increase as no new parts will be made available after a placement.

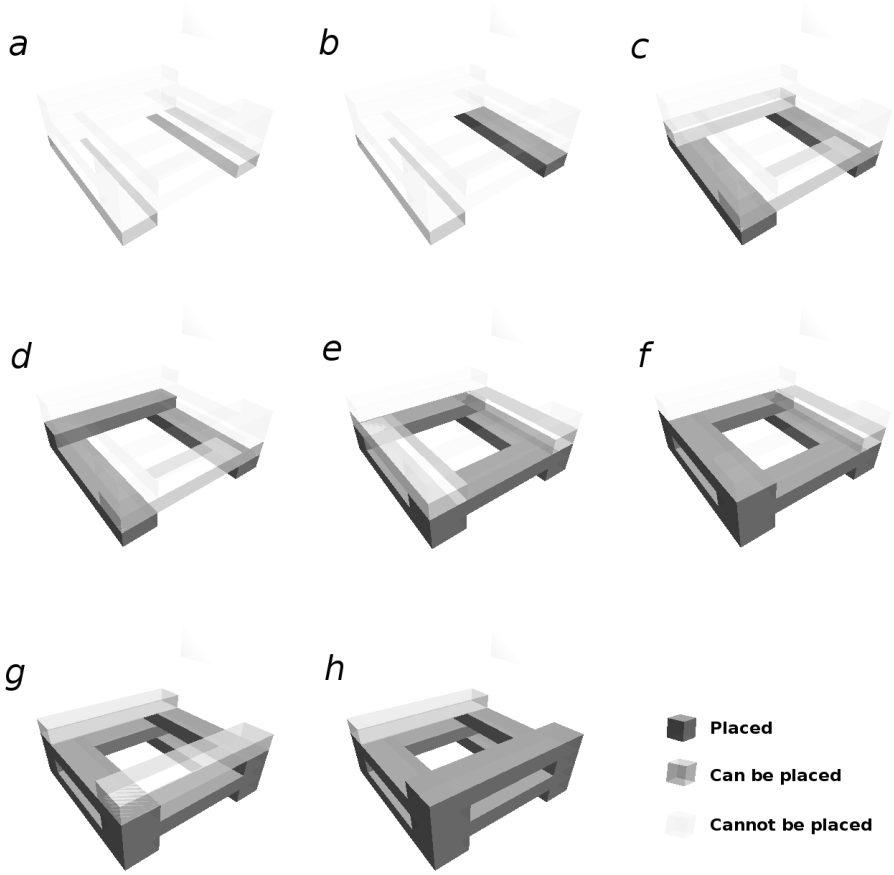


Figure 6-7: A rendering of system state over time. Images generated by the simulator from section 5.4 using the blueprint from the physical platform experiments.

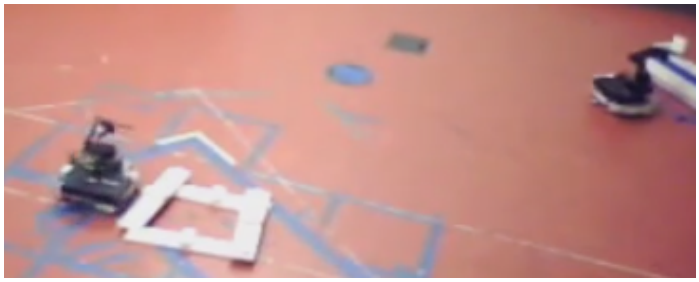


Figure 6-8: Even with feedback from vicon, our grippers introduce a fair amount of error. After a layer of pieces is placed (as in this figure) we will manually adjust the pieces to be more precisely located.

Shape	**Valid Placements**	**Manipulations** (Aided/Total)	Notes
4-part Box	4/4	3/16	
4-part Box	4/4	3/16	
4-part Box	4/4	4/16	
4-part Box	4/4	3/16	
4-part Box	4/4	4/16	
8-part Box	8/8	9/32	
8-part Box	8/8	8/32	
8-part Box	8/8	9/32	Replaced delivery robot battery mid-run
8-part Box	8/8	/32	2 delivery, 2 assembly
8-part Box	8/8	/32	2 delivery, 2 assembly
Wall	4/4	1/16	
Wall	4/4	5/16	
Wall	4/4	3/16	
Wall	4/4	4/16	

Table 6.2: Summary of Robot Assembly test runs using 1 delivery and 2 assembly robots unless noted otherwise.

Chapter 7

Extended Experiments

Using the same algorithms tested in 6, we repeated the experiments using new physical and software control systems. With the higher reliability. In these tables we also introduce the "utilization" metric, which measures the amount of experiment time each robot spends actively performing a task. We consider this a good proxy of the efficiency of our algorithms, where 100% utilization suggests for either class of robots suggests that either manipulation, navigation, or the relative populations of robot class are the highest-order constraint on our system.

7.1 Implementation

We replaced our roomba with mounted gripper platform with the KUKAyouBot (seen in figure 7-1). The robots are equipped with an internal computer running Ubuntu Linux, allowing for easy porting of our control software. We replaced all navigation and control modules from section 6-4 with equivalent packages for the KUKA-youbots, and implemented a handoff rather than passoff for the interaction between delivery and assembly robots. We augmented the youBots with WNCE2001 Wifi adapters, and migrated our messaging system onto the standard ROS messaging libraries.

Figure 7-1: Side view of the KUKA YouBot. The holonomic base allows for omnidirectional movement, while the five d.o.f. arm provides high-fidelity maniputation in the workspace around the robot

7.2 Experiments

7.2.1 Results (two robots)

This new task implemented the "box" design with six layers, as seen in figure 7-2. This new design introduced stronger stability and reliability constrainst from the experiments in the previous chapter, while maintaining similar ordering constraints. With only one partition, this experiment does not test the partitioning portion of our algorithms, making this largely a test of the transposition to a new physical system. All other processes follow from chapter 6. The assembly utilization hovered just over 80%, largely due to the distace between the depot and the construction site requiring long trips by the delivery robots.

Over our three trials there were no notable system failures and the struction was built without issues in stability, as outlined in table 7.1

Trial	**Runtime** (M:SS)	**Assembly utilization**	**Delivery utilization**	**Success** (Y/N)
1	19:32	83.1%	58.6%	Y
2	20:09	81.3%	69.7%	Y
3	22:17	84.5%	51.5%	Y
Avg	20:39	83.0%	59.9%	100%

Table 7.1: Summary of two-robot assembly trials for a tower.

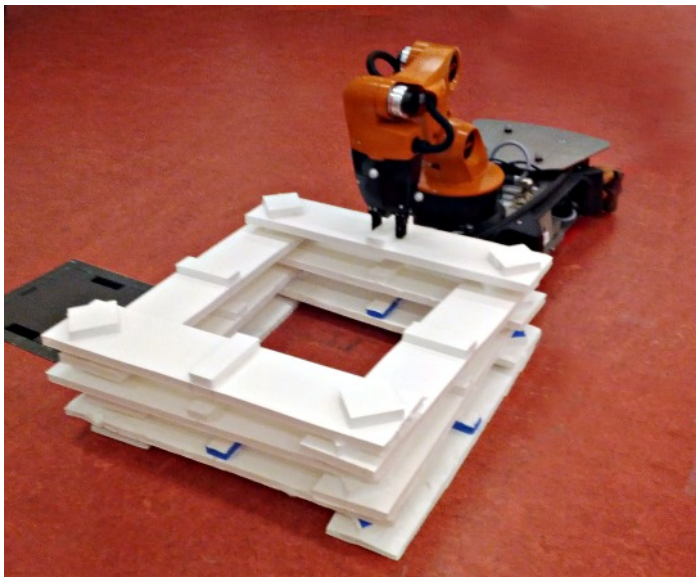


Figure 7-2: An assembly robot places the final part on the three-dimensional tower. The tower is composed of six layers of the box design simulated in figure 6-8. This tower is the result of trial #1 from Table 7.1.

Trial	**Runtime** (M:SS)	**Assembly 1 utilization**	**Assembly 2 utilization**	**Delivery 1 utilization**	**Delivery 2 utilization**	**Success** (Y/N)
1	9:16	84.8%	90.1%	95.4%	86.3%	Y
2	8:53	88.9%	87.3%	96.6%	96.1%	Y
3	8:46	89.9%	88.7%	93.0%	94.2%	Y
4	10:29	83.3%	86.8%	90.9%	94.4%	Y
5	12:34	87.8%	87.9%	82.0%	92.7%	Y
6	10:33	87.2%	87.9%	96.7%	93.9%	Y
7	8:42	89.7%	87.2%	97.9%	94.0%	Y
8	11:23	93.0%	84.9%	81.5%	93.7%	N
Avg	10:05	88.1%	87.6%	91.8%	91.9%	87.5%

Table 7.2: Summary of four-robot assembly trials for a tower.

7.2.2 Results (four robots)

Our final experiment reintruduces partitioning constraints to our new physical system. Over 8 iterations, we build the 6-layer "box" structure using two assembly and two delivery robots. The utilization was much higher across the board, and the runtime improved more than twofold. We suspect this is due to decreased waiting times with two delivery bots running simultaneously, despite a longer time per part placement but did not log metrics on these parameters as they are largely system specific. The results from this trial are outlined in Table 7.2

Chapter 8

Conclusion and Future Work

This thesis reviews work in distributed construction, experimentally explores existing work and identifies areas where extension would be most effective, and then provides those extensions by describing distributed algorithms for part placement and ordering which ensure that structures stay stable and reachable while under construction. It then demonstrates the effectiveness of those algorithms in practice.

8.1 Summary of Contributions

This work presents a system for describing distributed constuction tasks that takes into account the physical constraints of the structure and the construction system, provides algorithms for constuctruction within that constraint model, and demonstrates the efficacy of these algorithms in a

In chapter 3 we formalize a framework for considering construction as a series of discrete tasks tightly linked with several topological dependancies. This is a divergence from the more generally used methods of treating construction as either a single premade plan (ie, [16]) or a continuous field of "work" (ie, [2, 1]), and allows for a more direct mapping from physical systems to algorithms and proofs of correctness at the cost of heightened complexity.

In chapter 5, we re-constuct distributed construction algrithms to work under this more

accurate and constrained system. First we modified algorithms for equal mass partitioning to maintain correctness under desnsity functions with point-discontinuities. Next, using this partitioning function, we describe a class of functions that guartunees correctness and completability when possible in a physically constrained construction system, and describe a heuristic function that is a member of that class for use in experiments.

In chapter 4 we present an experimental platform for testing distributed construction algorithms and use it to test our foundational work by Yun et. al. Having demonstrated the efficacy of both our test platform and foundational work, the chapter 6 we test the algorithms defined in chapter 5.

Finally in chapter 7, we run our paritioning and assembly agorithms on higher-complexity structures with a homogenious team of robots. The authors of [29] use our work effectively to control a heterogeneous group of robots in their experimental setup.

8.2 Lessons Learned

Each increase in the resolution of a simulation introduces new complexities that are easy to overlook, and so the transitions between high level simulator down to a hardware platform provided many opportunities to discover potential problems with each algorithm. The actual transition to hardware introduced the problem of keeping hardware operational. Our first iteration required four batteries per robot and one per part, meaning to run an experiment one needed to check the charge of 24 batteries. It also used three different wireless communication protocol, an artifact of the constraints introduced by trying to run three thesis projects on one set of robots. With hundreds of cables, hand-soldered boards, and dozens of batteries, the chances of some part of the system failing dominated that of an algorithmic issue and made experiments a chore to prepare for. My second iteration used only 9 batteries, an improvement that allowed for several experiments to be run back to back, and we were able to meet some goals we had originally thought unattainable with our hardware. Striving for minimal complexity in all things makes loftier goals reachable.

8.3 Future Work

Work with HRI including skilled untrained laborers in the assembly process will also need to be explored before this platform is ready for practical implementation. Adapting to changing availability of different classes of parts and an extension to the algorithms presented in this thesis to avoid deadlocks waiting for parts with limited availability would also be an interesting exploration.

Our experiments solve the problem of observing location and structure state using an external localization system. Future work should consider both introduce a more realistic model for sensor constraint, and adapt not only for robot failure, but for unexpected external mutations in the target struction, such as from human intervention or damage to the structure during construction.

Finally intermediate structures (such as scaffolding or component construction) are often neccessary in complex construction tasks, and a fully general planner will need to introduce these non-permanent steps in construction into its planning algorithm.

Appendix A

Complexity of Equal-Mass Convex Partitioning of Pointmasses

One of the algorithmic contributions of this thesis is a distributed controller for partitioning clouds of point masses into equal mass convex partitions. This controller find locally optimal, rather than globally optimal, partitionings. We justify this by claiming that the task is NP- complete. The appendix presents a proof of that fact. The properties claimed for each widget can be confirmed with a simple brute force attempt at equal- mass partitioning.

We define Equal Mass Convex partitioning of point masses as follows: *given n point-masses, form $m < n$ partitions such that each partition has a total mass of $\frac{\sum_n mass(n)}{m}$ and is convex.*

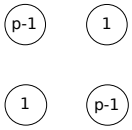
Lemma 4 *Equal Mass Convex partitioning of point masses is in NP*

Proof 4 *It is easy to see that verification can be done in polynomial time. Given a candidate partitioning, for each partition, compute the sum of the point masses (there are at most n) contained in that partition. If all partitions are equal then the partitioning is a correct partitioning, in all other cases it is not. Each point is visited once, giving a runtime of $O(n) \leq O(n^k)$.*

Lemma 5 *Equal Mass Convex partitioning of point masses is NP-Hard*

Proof 5 *We show this by reducing BSAT. The main intuition of this proof is that a decision of how to partition a subset of points can cause a chain reaction in the way partitions are assigned. We use these chain reactions as signals and describe how to build wires and logic gates out of sets of points. For convenience, I use N as shorthand for the mass of a wall-formed partition, and set m appropriately at the end such that $N = \frac{\sum_n mass(n)}{m}$. We also will only use pointmasses with positive mass in this reduction.*

Consider the four pointmasses below. Given an m of 2, there are only two possible valid partitionings. We could treat this as a Boolean variable, with one value assigned to each possible partition.

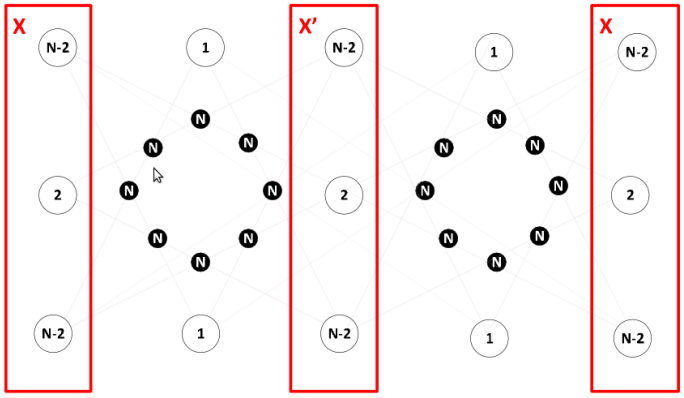


We can then build chains that carry that variables value. Effectively, a variable is represented by creating a situation where only one of two pointmasses will be available to include in a partition.

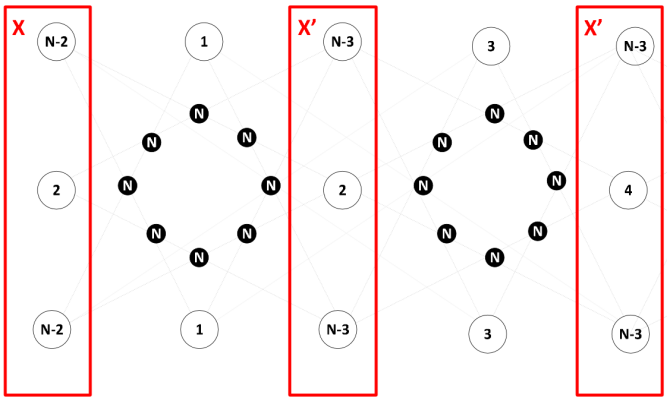
Because partitions must be convex, if three parts are co-linear, the outer two cannot be in the same partition unless the central point is as well. Because this reduction uses only pointmasses with positive mass, if the central point has mass N, the outer points cannot be in the same partition.

Using this property, we can construct an identity, crossover, NOT, and AND gates. Each gate takes the removal of one of each pair of "input pointmasses" as input, and omits one of the "output pointmasses" as output (assuming some other gate will use it as input). The correctness of each gate was confirmed by exhaustively constructing equal-mass partitions and testing for overlap in partitions.

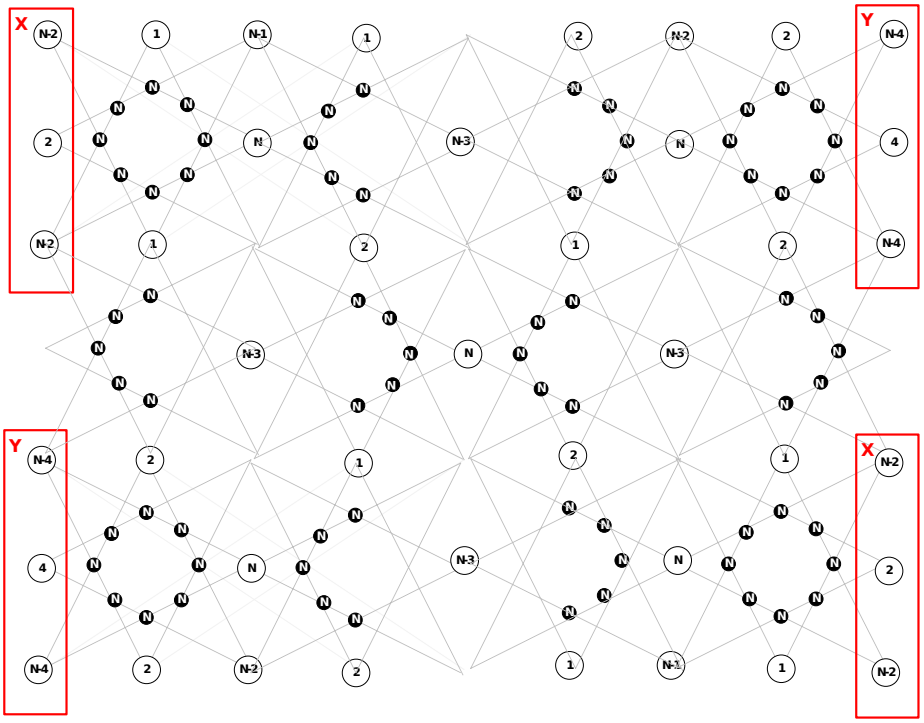
Identity



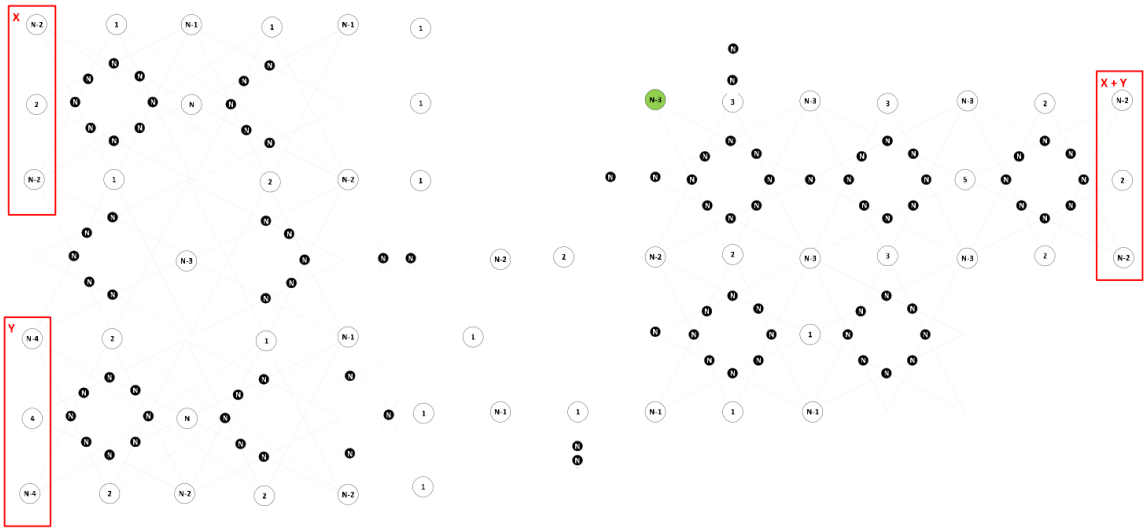
NOT



Crossover



AND



Having built these gates, we need to ensure that no partition spans more than one gate. We can do this by adding several pointmasses of mass N in the space between gates. For each pointmass in a gate, we add one such node between that pointmass and every point-

mass that is not in the same gate. If there are n gates, there are $O(n^2)$ such pointmasses added.

Using NOT and AND in series we can create NAND gates, with which we can build any Boolean circuit using $O(n^2)$ points broken into $O(n^2)$ partitions such that finding a equal-mass convex partitioning of the points would imply a solution to the Boolean Circuit. Therefore $BSAT \leq_p Equal - Mass\ Convex\ Partitioning\ of\ Pointmasses$

Theorem 6 *Equal Mass Convex partitioning of point masses is NP-Complete*

Proof 6 *From lemmas 4 and 5, Equal Mass Convex partitioning of point masses is both in NP and NP-hard. Therefore it is NP-complete.*

Bibliography

[1] S. kook Yun, D. A. Hjelle, H. Lipson, and D. 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.

[2] S. kook Yun, M. Schwager, and D. 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.

[3] H. Meyr, "Supply chain planning in the german automotive industry," *Or Spectrum*, vol. 26, no. 4, pp. 447–470, 2004.

[4] H. Mather and I. of Operations Management, *Competitive manufacturing*. Prentice Hall, 1988.

[5] "National census of fatal occupational injuries in 2009," News Release, August 2010.

[6] S. kook Yun, "Coordinating construction by a distributed multi-robot system," Ph.D. dissertation, MIT, 2010.

[7] "Automatic factory," Sep 1953. [Online]. Available: http://www.time.com/time/magazine/article/0,9171,818947,00.html

[8] C. Null and B. Caulfield, "Fade to black the 1980s vision of "lights-out" manufacturing, where robots do all the work, is a dream no more." Jun 2003.

[9] G. A. Fowler, "Holiday help: People vs. robots," *Wall Street Journel*, December 2010.

[10] P. Wurman, R. D Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," in *proceedings of the national conference on artificial intelligence*, vol. 22, no. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007, p. 1752.

[11] I. Hobkirk and R. Shecterle, "Scalability, flexibility, portability: Zappos re-writes the rules for warehouse design," Aberdeen Group, Tech. Rep., May 2008.

[12] S. Teller, M. Walter, M. Antone, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J. How, A. Huang, *et al.*, "A voice-commandable robotic forklift working alongside humans in minimally-prepared outdoor environments," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 526–533.

[13] Robots using ros. [Online]. Available: http://www.ros.org/wiki/Robots

[14] F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*, ser. Applied Mathematics Series. Princeton University Press, 2009, electronically available at http://coordinationbook.info.

[15] G. Theraulaz and E. Bonabeau, "Coordination in distributed building," *Science*, vol. 269, no. 5224, pp. 686–688, 1995. [Online]. Available: http://www.sciencemag.org/content/269/5224/686.abstract

[16] J. Werfel and R. Nagpal, "International journal of robotics research," *Three-dimensional construction with mobile robots and modular blocks*, vol. 3-4, no. 27, pp. 463–479, 2008.

[17] L. Matthey, S. Berman, and V. Kumar, "Stochastic strategies for a swarm robotic assembly system." in *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 1953–1958.

[18] J. Thomas and K. Baker, "Modelling of assembly partitions for a distributed environment," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*. IEEE, 1993, pp. 52–57.

[19] S. kook Yun and D. Rus, "Optimal distributed planning for self assembly of modular manipulators," in *Proc. of IEEE/RSJ IEEE International Conference on Intelligent Robots and Systems*, Nice, France, Sep 2008, pp. 1346–1352.

[20] A. Bolger, M. Faulkner, D. Stein, L. White, S. kook Yun, and D. Rus, "Experiments in decentralized robot construction with tool delivery and assembly robots," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[21] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *Robotics and Automation, IEEE Transactions on*, vol. 20, no. 2, pp. 243–255, 2004.

[22] M. Schwager, J.-J. Slotine, and D. Rus, "Decentralized, adaptive control for coverage with networked robots," in *Robotics and Automation, 2007 IEEE International Conference on*, april 2007, pp. 3289–3294.

[23] M. Pavone, E. Frazzoli, and F. Bullo, "Distributed policies for equitable partitioning: Theory and applications," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, dec. 2008, pp. 4191–4197.

[24] S. kook Yun and D. Rus, "Distributed coverage with mobile robots on a graph Locational optimization and equal-mass partitioning," in *Workshop on the Algorithmic Foundations of Robotics*, 2010.

[25] S. Yun and D. Rus, "Adaptation to robot failures and shape change in decentralized construction," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. Institute of Electrical and Electronics Engineers, 2010.

[26] M. Hsieh and J. Rogoff, "Complexity measures for distributed assembly tasks," *Proc. of the 2010 Performance Metrics for Intelligent Systems Workshop*, 2009.

[27] P. Yiu, "The uses of homogeneous barycentric coordinates in plane euclidean geometry," *International Journal of Mathematical Education in Science and Technology*, vol. 31, pp. 569–578, 2000.

[28] F. Preparata and S. Hong, "Convex hulls of finite sets of points in two and three dimensions," in *Communications of the ACM*, 1977.

[29] J. R. Ross A. Knepper, Todd Layton and D. Rus, "Ikeabot: An autonomous multi-robot coordinated furniture assembly system," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2013.

[30] T. R. Schoen and D. Rus, "Decentralized robotic assembly with physical ordering and timing constraints," in *Proceedings of the IEEE/RSJ International Conference on Intellegent Robots and Systems*, 2013 (submitted).

[31] T. R. Schoen, "Constraint-aware distributed robotic assembly and disassembly," Master's thesis, MIT, 2012.

[32] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

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

[34] M. Nechyba and Y. Xu, "Human-robot cooperation in space: SM^2 for new spacestation structure," *Robotics & Automation Magazine, IEEE*, vol. 2, no. 4, pp. 4–11, 1995.

[35] D. Stein, T. R. Schoen, and D. Rus, "Constraint-aware coordinated construction of generic structures," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011 (submitted).

[36] G. Haag, "Automated parking garage," US Patent US Patent App. 20,040/258,506, 2004.

[37] M. Pavone, E. Frazzoli, and F. Bullo, "Distributed algorithms for equitable partitioning policies: Theory and applications," in *IEEE Conference on Decision and Control*, Cancun, Mexico, Dec 2008.

[38] O. Baron, O. Berman, D. Krass, and Q. Wang, "The equitable location problem on the plane," *European Journal of Operational Research*, vol. 183, no. 2, pp. 578–590, 2007.

[39] K. Robotics. Kuka youbot manual v0.81. [Online]. Available: http://youbot-store.com/downloads/

[40] M. Faulkner, "Instrumented tools and objects: Design,algorithms, and applications to assembly tasks," Master's Thesis, Massachusetts Institute of Technology, CSAIL Distributed Robotics Laboratory, June-Aug. 2009.

[41] 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.

[42] S. kook Yun, D. A. Hjelle, H. Lipson, and D. 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.

[43] C. Detweiler, M. Vona, Y. Yoon, S. kook Yun, and D. Rus, "Self-assembling mobile linkages," *IEEE Robotics and Automation Magazine*, vol. 14(4), pp. 45–55, 2007.

[44] "irobot create open interface (oi) specification." [Online]. Available: iRobot.com

[45] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[46] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, "Visibility of disjoint polygons," *Algorithmica*, vol. 1, no. 1, pp. 49–63, 1986.

[47] N. Koenig and A. Howard, "Gazebo-3d multiple robot simulator with dynamics (2003)."

[48] B. Chu, K. Jung, Y. Chu, D. Hong, M. Lim, S. Park, Y. Lee, S. Lee, M. Kim, and K. Ko, "Robotic automation system for steel beam assembly in building construction," in *Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on*. IEEE, 2009, pp. 38–43.

[49] V. Hunt, *Industrial robotics handbook*. Industrial Press Inc., 1983.