# Distributed Assembly Strategies for Teams of Autonomous Robots 

A Thesis<br>Submitted to the Faculty<br>of<br>Drexel University<br>by<br>Joshua W. Rogoff<br>in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering
June 2011
(C) Copyright 2011

Joshua W. Rogoff. All Rights Reserved.

## Acknowledgments

Firstly, I would like to thank my advisor, Dr. Ani Hsieh. I learned so much by working with her, and without her support and guidance, I would not have been able to produce this document. Having the opportunity to do research in the SAS lab was thrilling for me, and I truly enjoyed my time there. I would also like to acknowledge Bill Mather and James Worcester. They were a huge help and I enjoyed working with them on a daily basis. Finally, I want to thank my mother and father for always supporting me, my brother for providing necessary distractions, and Lauren for her constant love and support.

## Contents

List of Tables ..... vi
List of Figures ..... vii
Abstract ..... x
1 Introduction ..... 1
1.1 Problem Statement ..... 2
1.2 Related Work ..... 2
1.3 Organization of this Work ..... 4
2 Distributed Assembly Testbed ..... 5
2.1 Assembly Materials ..... 5
2.2 Hardware ..... 7
2.2.1 iRobot Create ..... 7
2.2.2 Lynxmotion Arm ..... 7
2.2.3 Hokuyo Laser Rangefinder ..... 9
2.3 Software ..... 9
2.3.1 Robot Control ..... 10
2.3.2 Arm Control ..... 11
2.3.3 Block Extraction ..... 12
2.4 Robot Navigation ..... 13
2.4.1 Overhead Localization System ..... 14
2.4.2 Hardware Implementation ..... 15
3 Motivating Example ..... 19
3.1 One Robot Building a 2-D Structure ..... 19
3.1.1 Running the Simulation ..... 20
3.2 Two Robots Building a 2-D Structure ..... 21
4 Distributed Assembly in 2-D ..... 23
4.1 Methodology ..... 23
4.1.1 Scoring Cells Adjacent to the Structure ..... 23
4.1.2 Different Assembly Techniques ..... 24
4.2 Experimental Analysis ..... 29
4.3 Results ..... 30
4.3.1 Simulations ..... 30
4.3.2 Experiments ..... 31
4.4 Discussion ..... 32
5 Decentralizing the Assembly Tasks ..... 35
5.1 Equal Mass Decomposition of Target Structures Using Voronoi Diagrams ..... 35
5.2 Maintaining Navigation Awareness ..... 38
5.2.1 Weighted Mass-Based Partitioning ..... 38
5.2.2 Defining an Optimization Strategy for Navigation-Aware Par- titions ..... 39
5.2.3 Using the Brushfire Algorithm to Partition Target Structures ..... 41
5.3 Defining Structural Complexity ..... 44
5.3.1 Problem Definition ..... 45
6 Methods for Assembly of Partitioned Structures ..... 49
6.1 Experimental Setup ..... 50
6.2 Experimental Results ..... 51
7 Concluding Remarks ..... 53
7.1 Future Work ..... 53
Appendices ..... 56
A Differential Drive Robot Kinematics for Closed-Loop Navigation ..... 56
B The A* Algorithm ..... 58
C Voronoi Decompositions ..... 60
D The Brushfire Algorithm ..... 64

## List of Tables

4.1 Results for distance traveled by robot for each assembly method givenfour different shape-maps.324.2 Results for distance traveled by robot for each assembly method. The robot traveled an average of $9.2 \%$ farther for the perimeter control method. ..... 32
5.1 Mass distribution across the Voronoi partitions for the structures shown in Figures 5.1(b) and 5.2. ..... 38
5.2 Mass distribution across the navigation-aware partitions $T_{i}$ for the structures shown in Figure 5.9 where $\phi_{i}=\phi\left(T_{i}\right)$. The boundaries for $T_{i}$ are denoted by the thicker lines. ..... 45
5.3 Complexity measures for various target structures ..... 48
6.1 Experimental results for 2 assembly sequences of 3-D structure ..... 52
6.2 Experimental results for 2 assembly sequences of 3-D structure ..... 52

## List of Figures

$2.1 \mathrm{M}^{3}$ robot ..... 6
2.2 Acrylic block used in assembly ..... 6
2.3 Intelliblock prototype ..... 7
2.4 A polar plot depicting data from an LRF reading ..... 102.5 Laser scan data with line fitting and block detection. The image showstwo blocks side by side, which indicates a structure.132.6 Map of the workspace with the coverage area of each camera and eachcamera's approximate location. The cameras are named north, south,east and west14
2.7 Position of the robot as determined by the overhead localization system (in blue) and the odometry sensors (in red). ..... 16
3.1 The robots are represented by the green dots with arrows markingtheir orientations. The black square around the robot indicates thatthe robot is carrying a block. The red square represents $\mathbf{m}$ and the bluesquares represent $C_{i}$ where the blocks are obtained. The yellow squaresrepresent $\mathcal{S}$, and the black squares around the yellow ones representplaced blocks. This figure shows four screen shots of the simulation as
it is happening. ..... 22
4.1 Neighborhood of 4, seen in (a), and neighborhood of 8, seen in (b) . . ..... 24
4.2 Depiction of the scoring simulation: The empty squares represent $\mathcal{S}$.The symbols on the blocks adjacent to the structure indicate the scorefor those spots. The individual scores can be seen in the figure. Thearrival site $h$ is depicted by a green square, and the side that the robotarrives at is indicated by the triangle.25
4.3 Experimental setup for scoring experiment ..... 29
4.4 Photos of experiment taking place. ..... 30
4.5 The four shape-maps evaluated in this simulation ..... 31
4.6 Plot of the average distance traveled for each block placement over 50 iterations of each assembly method ..... 34
5.1 Voronoi decomposition of a simple square structure with uniform mass. The shaded region represents the structure, the black dots represent the Voronoi points and the lines represent the edges. ..... 37
5.2 Four maps with their final equal-mass Voronoi cells. ..... 38
5.3 Different weighting schemes applied to the B map. ..... 39
5.4 Six shapemaps with the brushfire algorithm applied to the outer edge of each structure and map border. ..... 41
5.5 Six shapemaps with the brushfire algorithm applied to the inside of each structure. ..... 42
5.6 Six shapemaps with $V_{R}$ shown with blue and red dots and $V_{S}$ shown with yellow dots. ..... 42
5.7 Navigation-aware partitioning of structures using $V_{S}$ as Voronoi points. ..... 42
5.8 Decomposition using $s$ nodes as boundaries for cells. ..... 43
5.9 Navigation-aware task partitioning where the boundaries for $T_{i}$ are denoted by the thicker lines. ..... 44
6.1 Image of the target structure for experiment ..... 50
6.2 Two different assembly sequences - blocks shown in blue are assembledby Robot 1 and blocks in yellow are assembled by Robot 2. . . . . . 50
6.3 Diagram of the experimental setup ..... 51
6.4 Two robots assembling the structure during sequence 1 experiment ..... 52
A. 1 Closed loop control diagram ..... 57
B. 1 An example solution to $A^{*}$. The heuristics are the $h_{i}$ scores for each spot. [3] ..... 59
C. 1 An example of a Voronoi diagram ..... 61
C. 2 Fortune's Sweepline Algorithm ..... 63D. 1 Brushfire algorithm applied to the outer border of a structure and theedges of the map.64


#### Abstract

Distributed Assembly Strategies for Teams of Autonomous Robots Joshua W. Rogoff M. Ani Hsieh, Ph.D.


The distributed assembly problem involves using a team of robots to assemble a structure autonomously. The goal is to develop a strategy such that the robots assemble the structure correctly and in the most efficient way possible. This thesis outlines different single robot assembly strategies, different methods for partitioning the building tasks amongst multiple robots, and defining the complexity of a structure to be assembled. The scope of work includes investigating different assembly strategies through design and analysis of different assembly algorithms, developing simulations to evaluate and validate the different assembly strategies, comparing the proposed methods with existing approaches, and implementing selected assembly strategies on an actual robotic testbed.

## Chapter 1

## Introduction

With every passing day, we see robots effecting our lives more and more. These robots aim to make certain aspects of our lives easier, whether it is a vacuum cleaner in our home, or an automated warehouse that allows a company to deliver products more quickly and cost effectively to our doorstep. It is with this knowledge that we can understand how a team of robots cooperating to complete a task is not a far-fetched idea.

Distributed assembly aims to utilize a multi-robot team that works together to assemble a structure. Real-world applications of the distributed assembly idea can range from building everyday structures, to assembling a structure underwater or on another planet. Although a common structure such as an office building or house may not require a team of robots to assemble, imagine a group of astronauts arriving at the moon to an already assembled base, or a team of robots being able to patch a hole in a damaged nuclear plant. These scenarios present applications where it would be either too hazardous, or simply impossible for humans to complete the task. It is with these examples that the usefulness of distributed assembly research can be understood.

### 1.1 Problem Statement

Distributed assembly introduces several problems. Tasks that must be completed include coordinating a team of robots to work cohesively and effectively, dividing the workload amongst the team, developing different assembly strategies, and testing these strategies through simulation and experimentation. Assembly can be completed in either a centralized or decentralized manner, using homogeneous or heterogeneous robots. The problem we aim to solve involves outlining methods for decentralized multi-robot assembly given a team of homogeneous robots.

### 1.2 Related Work

There are many conceivable ways to tackle the distributed assembly problem. The type of hardware being used, coupled with the vast algorithmic possibilities allow for countless combinations of solutions. When designing solutions for a specific suite of hardware, limitations can play a role in the way the algorithm is structured. In [21] and [22], the dynamics and size of the robot being used effect the algorithm. In these cases, a row of blocks must be completed (in accordance with the predefined structural shape) once it is started, because the robot being used for the experiment is not capable of filling in a gap in the structure once it has been created. In [14] Sanderson et al. introduce an assembly sequence planning algorithm using a structure that has various attachment constraints.

Werfel et al. explore the use of intelligent building materials in [22, 23]. These building materials have varying capabilities, but all maintain the general idea that when a robot arrives at the structure, the structure is able to communicate information about itself. This work is tested using a Lego construction robot described in [15]. In [10], Klavins introduces a robotic building block system that is capable of self
assembly. In other words, the blocks act as both the assembly robot and the building material. Similarly, Yim et al. describe their modular self-assembling robot in [26]. This introduces the idea that building blocks can gain the ability to assemble into a body that can become mobile.

Three-dimensional assembly approaches are also explored. Terada et al. present an automatic modular assembly system (AMAS) capable of 3-d assembly, which includes modular building materials and a robot capable of assembling a 3-d structure in [17]. Werfel et al. present a more abstract approach which focuses on assembly strategy rather than hardware implementation in [24].

Division of labor is another aspect of distributed assembly that must be addressed. Cortes et al. describe a way of using a continuous time Lloyd algorithm to drive a team of robots to positions such that each robot has the same amount of work to do in [4]. This algorithm is put to use to divide structures into equal-mass partitions amongst a team of robots in [28]. Yun et al. use their algorithm from [28] to develop an algorithm to account for robot failure during construction in [27]. This algorithm was also used by Pimenta et al. in [12] to develop a coverage controller for real time tracking. Schwager also uses Voronoi decompositions to develop a decentralized and adaptive control algorithm for coverage in networked robots in [16].

Voronoi diagrams, although effective, are only one way to partition the assembly tasks. Another way to approach the problem is to define complexity measures for the assembly tasks. In [13], Sanderson describes a "parts entropy" method for robotic assembly. This method can be used to quantify aspects such as assembly procedure and flexibility of an assembly system. Sanderson's work is put to use in [11], where the parts entropy method is used to measure the complexity of a system of self-replicating robots.

Many of the ideas used in distributed assembly relate to other multi-robot appli-
cations. Hsieh presents control strategies for teams of networked robots in [7] and [6]. Methods for simultaneous localization and mapping (SLAM) for a robot swarm is outlined by Thrun in [18]. In many of the above works, it is assumed that the map used by the robot team to navigate is given a priori. SLAM is very useful if this map is not known or only partially known.

### 1.3 Organization of this Work

The remainder of this thesis is organized as follows. Chapter 2 describes the distributed assembly testbed used to carry out experiments. This includes the robots, the software used to control the robots as well as the building materials used for assembly. A motivating example is explored in Chapter 3, which describes a simple assembly example and explains the challenges of distributed assembly. In Chapter 4, methods are introduced to distribute the assembly tasks by scoring legal block attachment sites. Chapter 5 explores different ways to decentralize the assembly tasks and define structural complexity. After describing ways to divide the workload in Chapters 4 and 5, a method for structural assembly is described in Chapter 6. Chapter 7 concludes the work with a summary and direction for future work.

## Chapter 2

## Distributed Assembly Testbed

Throughout this work, we will be presenting experimental results. These results were achieved using experiments designed to utilize the mini-mobile manipulator platform ( $M^{3}$ for short). The experiments were carried out in the Scalable Autonomous Systems lab (SAS lab) at Drexel University. Each $\mathrm{M}^{3}$ robot consists of an iRobot Create programmable robot base, a 6-degree-of-freedom (DOF) Lynxmotion AL5D robotic arm, and a Hokuyo URG-04LX-UG01 scanning laser rangefinder (LRF). A picture of an $\mathrm{M}^{3}$ robot is found in Figure 2.1.

Each $\mathrm{M}^{3}$ robot is assigned its own ACER Ferrari laptop. These laptops allow the user to easily interface with the robots and control each component. The software for the $\mathrm{M}^{3}$ testbed was written in Matlab and consists of functions and scripts which reside in toolboxes dedicated to each piece of hardware of which the $\mathrm{M}^{3}$ consists.

### 2.1 Assembly Materials

The structures that the $\mathrm{M}^{3}$ robots assemble are made out of white opaque acrylic blocks. The acrylic was cut into pieces using a laser cutter and taped together to create the block shape. The shape of the blocks resemble Lincoln-Logs, and have


Figure 2.1: $\mathrm{M}^{3}$ robot
grooves that allow them to interlock. An image of a building block is found in Figure 2.2. These blocks allow us to design 3-D structures for the robots to assemble.


Figure 2.2: Acrylic block used in assembly

In some of our simulations, we make the assumption that the building material has some processing and communication capability. In experimentation, we do not use these "intelligent building materials", however, they are feasible. Figure 2.3 shows a prototype of an intelliblock fabricated out of balsa wood. This block has contact sensors to determine if and where other blocks are attached to it, and can communicate the information to a robot by blinking an LED.


Figure 2.3: Intelliblock prototype

### 2.2 Hardware

### 2.2.1 iRobot Create

The iRobot Create is based on iRobot's Roomba vacuum cleaner. It has a cargo bay which is used to store excess wires and the battery for the arm, and threaded holes which are used to mount the arm and LRF. The robot has a suite of sensors that include a cliff sensor, wall following sensor, bump sensor, odometry sensors, lift sensor, among others. For our purposes, we only need to utilize the odometry sensors for closed loop navigation and to determine distance and angle traveled. The robot has a USB cable that allows it be programmed and to communicate with the computer. It is powered by a rechargeable NiMH battery.

### 2.2.2 Lynxmotion Arm

The arm consists of 6 servo-motors (the base, shoulder, elbow, wrist, wrist rotation, and grip), and linkages that connect them. The motors are controlled by one SSC-32 servo controller which interfaces with the computer via USB. The servos accept pulse lengths in milliseconds, which determine the angle that the servo will go to and speeds in pulselength/second as input commands from the controller. The arm and motor controller are powered by one 7.2 V battery, which connects to the controller board. A switch on the board allows for easy power on and off.

Each time the arm is powered on, it must be sent to an initial, or home position.

The first movement of each servo once powered on is the servo's maximum speed, so it is important that the arm makes this motion to a position that won't damage the robot or anything in its direct vicinity. Since no two robotic arms are identical, each arm was calibrated to obtain the equations to convert the desired angle for each servo to a pulse length. For some of the servos, the manufacturer's recommended equations work fine, but for others, a different calibration technique is needed.

To calibrate a servo, we first set up the arm in a position where the base will not move. We then set up a system such that we can measure the angle that the servo goes to in degrees. Once this is achieved, the process is simply to send a pulse length to the servo being calibrated, and the angle that the servo goes to must then be measured. This is done for the full range of motion for the servo. Once complete, a simple linear regression is done to determine the two constants, $\alpha$ and $\beta$ in

$$
\begin{equation*}
p=(\alpha \theta)+\beta, \tag{2.1}
\end{equation*}
$$

where $p$ is the pulse length and $\theta$ is the desired angle in degrees. Calibration of the servos is important, because with a 6 -DOF arm, if one of the lower servos (such as the shoulder or elbow) goes to the wrong angle, that error propagates until it reaches the end-effector. This would cause the arm to miss the target location.

Once calibrated, each arm is capable of being given a desired end-effector position. The robot uses inverse kinematics to determine the angle of each servo to achieve this desired position. Since the end effector could be at the same final position but be in different configurations, the gripper width, wrist rotation angle, and elbow angle must also be explicitly defined.

### 2.2.3 Hokuyo Laser Rangefinder

A laser rangefinder is a device that uses a laser to obtain distance measurements. Like the other hardware, the LRF interfaces with the computer via USB. It also receives its power from the computer and does not need to be powered on or off. The device provides distance readings over a $240^{\circ}$ field of view, with a $0.35^{\circ}$ resolution. Figure 2.4 shows an example of a polar plot containing distance readings from the LRF. It is with this device that we locate precise block positions so that the robot may assemble the structure. Since we know the size of the blocks precisely, and the LRF gives us distance measurements, the robot is able to determine if what it is seeing is a block or another obstacle.

We do not use the LRF to navigate around the entire workspace, since this would take a very long time. Instead, we use this device once we are reasonably close to a block or structure, and use it to precisely navigate to a position such that a block can be picked up or placed using its data.

In order to pick up a block, the robot must drive up to the block to a position and angle such that the arm is capable of picking up the block once the robot has stopped. To place a block on the structure, there must be a block attached to the structure that the robot can use for position reference. The robot must recognize the location and position of the structure, and be aware of the height at which the block must be placed. The LRF is not responsible for knowing the height, since it only capable of doing a 2-D scan.

### 2.3 Software

To synthesize the different components of the $\mathrm{M}^{3}$, a set of toolboxes have been written in Matlab. Each component is connected to the computer using USB, and when each


Figure 2.4: A polar plot depicting data from an LRF reading
component is initialized, a serial object is created so that commands can be sent and data can be received.

### 2.3.1 Robot Control

The Create's Matlab toolbox consists of commands that are able to read the various on board sensors and control the motors. The robot can be driven using one of several different commands, which drive the robot by giving the wheels individual speed commands, give the robot a desired forward velocity and angular velocity, or set a forward speed and desired turning radius. We use the function that provides each wheel with its own velocity, since these values are calculated in the closed-loop navigation algorithm outlined in Appendix A.

To determine the distance and angle traveled by the robot, we initialize the sensor before each trip to or from the structure. This initialization zeros out the data and allows us to eliminate any error that may have built up over a previous trip. This data can then be recorded and used for experimental results.

### 2.3.2 Arm Control

Control of the robotic arm is open loop. For this reason, it is important that the calibration constants are correct because no error correction can be made during motion. The lowest level functions in this toolbox send serial commands to the individual servos which include the servo port number, the pulse length and the speed. There is one function to control each servo on the arm. Within each of these functions lies the equation which converts degrees to pulse lengths. These equations are where the calibration constants come into play. This allows us to define the position we want the servo to go to (in degrees). These functions also check to make sure that the desired angle is within the working range for each servo. Other important lower level functions that are included in the toolbox are a function which combines all of the servo commands to move them in one fluid motion, and an inverse kinematics function which calculates each servo angle based on the desired position and orientation of the end effector.

The lower level functions are put to use in the functions that send commands to pick up and place the blocks. The main inputs to both of these functions are the block's $x, y, z$, and $\theta$ values, which are handed over from the LRF data. There are several steps taken to pick up and place blocks which are necessary because the motion of the arm is open loop. To pick up a block, the arm is first moved to a position just above the block, and the gripper moves to its fully open position. The arm then slowly lowers itself straight down to the point where it is able to grip the block, but is not yet touching it. The arm then grips the block, and raises itself straight up to ensure proper clearance from anything the block may have been resting on. Once the arm and block are at a safe distance from any obstacle, the arm returns to the home position holding the block.

The function to place down a block is very similar, in that it contains the same
steps in reverse order. The one small difference is that instead of placing the block directly back onto the ground or structure, it lowers it to a position such that the block must drop a short distance. This is because the blocks are designed to fall into place.

### 2.3.3 Block Extraction

The laser rangefinder needs only one command, which sends a signal to the device to take a reading, and then receives that data. This data is simply a $685 \times 2$ vector containing the angle in the first column and the distance reading in the second. The challenging part is interpreting the data once it has been received. The main purpose of the LRF is to detect blocks in the vicinity of the robot, and this data is then used to drive the robot to a position such that the arm is close enough to carry out its manipulation.

Block detection is done using a line fitting algorithm. For sections of data that appear to be linear, we ask whether or not it could be a block, given the length of the line and the distance from the robot. Once all of the blocks in the space are found, the algorithm calculates the centroid of each line. The robot then determines which block is closest, and drives to it. An image of the laser data after the block detection algorithm has been run can be found in Figure 2.5. This figure shows two blocks side by side, which we know to interpret as a structure.

When using the LRF data to drive to a block, the robot calculates the distance and angle of the block with respect to the robot. Once this data is obtained, the robot drives forward a small amount and takes another reading. The duration that the robot drives is proportional to the distance it is from the block. When the robot is close enough to the block, it will stop and communicate the block's position and orientation to the arm, which will then pick up or place the block.


Figure 2.5: Laser scan data with line fitting and block detection. The image shows two blocks side by side, which indicates a structure.

### 2.4 Robot Navigation

With all of the hardware explained, we now wish to define the navigation strategy. The robot is able to maneuver around the workspace by combining data from the lab's overhead localization system (OLS), using data from the Create's odometry sensors, and using data from the LRF when close to blocks. We will focus on the first two sensing mechanisms, which are used when the robot is traveling from one waypoint to another and not trying to pick up or place a block. In the first iteration of our controller, we only use the odometry sensors when the OLS data is unreliable (further described in Subsection 2.4.2. This controller is used to navigate the robots in the experiment described in Section 6.1 and Section 4.2. The second iteration implements an extended Kalman filter (EKF) to help better estimate the robot's actual position and conbine data from both sensors for every time step. The closed-loop navigation that is implemented in both cases is described in-depth in Appendix A.

### 2.4.1 Overhead Localization System

The workspace in the lab consists of a flat rectangular floor. This floor acts as a Cartesian coordinate system with $x$ and $y$ axes in the middle. These axes allow us to determine the precise locations of each robot in the space. Four cameras, one on each side of the rectangle, are set up such that the entire space is in view of at least one camera. The coverage map can be seen in Figure 2.6.


Figure 2.6: Map of the workspace with the coverage area of each camera and each camera's approximate location. The cameras are named north, south, east and west.

To locate a robot in the space, the cameras report every orange rectangle, or blob that they see, and the blob's pose (position and orientation). Each robot has one orange rectangle attached to it, as seen in Figure 2.1. When there are multiple robots in the space, the system simply returns an array of poses for each blob seen. This system runs on a computer connected to the lab's network, and transmits the data
to any other computer on the network requesting that data.

### 2.4.2 Hardware Implementation

The closed loop navigation strategy, which allows the robot to navigate around the workspace, is described in Appendix A. To implement the controller, we will use both the overhead localization system and the odometry sensors on the Create. As previously stated, OLS relies on four cameras to cover the entire space. Although there is some overlap between the cameras, as seen in Figure 2.6, there are large portions of the map that are only covered by one or two cameras. Due to this lack of redundancy, the situation often arises where the arm of the robot prevents some or all of the orange blob from being visible to one or multiple cameras. Additionally, problems can arise when the robot is near the boundary of one camera's range. There are also instances where the robot's arm causes one blob to appear as multiple blobs to the camera or cameras.

Ideally, we would like to be able to take the OLS information as ground truth. Although there is some error associated with each reading from the cameras, this error does not propogate as time goes on (unlike the odometry sensors). As a result, when we can determine that the reading recieved from the OLS is reliable (i.e. a reasonable pose compared to the previous pose), we do take the OLS information to be ground truth. Figure 2.7 shows the robot's position according to the OLS data and the odometry data. We see that as the robot drives, the odometry data becomes farther from the robot's actual position.

In order to implement the control law on our robot, all we need is the current position and orientation of the robot, and the desired final position and orientation. From the OLS, we may be receiving multiple coordinates if more than one robot is visible to the system. To determine which blob position is ours, we must first define


Figure 2.7: Position of the robot as determined by the overhead localization system (in blue) and the odometry sensors (in red).
the robot's starting position explicitly. The blob closest to this position is remembered as the correct one. The odometry also uses this position to be the robots position, unless the blob is not visible to the OLS, in which case the known starting position is used.

The OLS is capable of recognizing a blob, and giving the blob's orientation, however, because the blob is just a rectangle, there is no way for it to know which way is forward or backward. To resolve this issue, we drive the robot forward for a short distance and recalculate the position. Using the old position and the new position, we can construct a vector and determine which direction is forward, and whether or not the OLS guessed correctly the first time. Once this direction is determined, we can use it to check whether or not the OLS angle information is flipped every time the robot receives information.

## Kalman Filtering for Navigation

To improve on the navigation strategy, we implement an extended Kalman filter (EKF) to analyze localization information fed to the robot from the sensors. A standard Kalman filter uses the sensor information available to determine where it believes the robot's position is at a given time, or $\operatorname{bel}\left(x_{t}\right)$ at time $t$. First, the robot determines where it thinks it is based on the robot's specific dynamic model, which is the behavior of the state vector $[x, y, \theta]^{T}$. The position estimation is then updated using the observation model, which is the relationship between the measurements and the state vector. To generate the observation model, data is gathered a priori to determine the sensor output's relationship to the robot's state. What results is a calculation of the mean $\mu_{t}$ and covariance $\Sigma_{t}$ of the robot's position at time $t$.

The EKF is based on the Kalman filter described above. The standard Kalman filter assumes that the system is linear yet this is rarely the case in the real world and our system is no exception. To account for these nonlinearities, additional steps are added to the standard Kalman filter to linearize the system. The EKF algorithm is described in Algorithm 2.1.

Our system uses the $\mathrm{M}^{3}$ 's odometry sensors and the OLS information to generate the observation model. As mentioned before, we take the OLS data to be ground truth. By using the EKF, we are able to combine the OLS data and the odometry data (instead of having to choose between one or the other), and when the OLS data is unavailable or unreliable, the filter will still ensure a better pose estimation than a single bad data point [19, 2, 9].

```
Algorithm 2.1 Extended Kalman Filter
    Linearize the motion model by computing Jacobian matrices \(G_{t}\) and \(V_{t}\)
    Determine the motion noise covariance matrix \(M_{t}\)
    3: Calculate the predicted pose of the robot since its last movement \(\mu_{t}\)
    Use \(G_{t}, V_{t}\) and \(M_{t}\) to compute the uncertainty skirt \(\Sigma_{t}\)
    for \(i=1 \rightarrow 2\) do
        define noise in sensor reading to \(Q_{t}\)
        calculate the predicted measurement based on sensor data \(z_{t}^{i}\) and Jacobian \(H_{t}^{i}\)
        determine uncertainty corresponding to the measurement \(S_{t}^{i}\)
        compute the Kalman gain \(K_{t}^{i}\)
        use \(K_{t}^{i}\) and \(z_{t}^{i}\) to update the pose estimate \(\mu_{t}\)
        use \(K_{t}^{i}\) and \(H_{t}^{i}\) to update the uncertainty in the pose \(\Sigma_{t}\)
    end for
```


## Chapter 3

## Motivating Example

In order to understand how to confront the problem of distributed assembly, we devise a simple simulation. The purpose of this is to illustrate the challenges that distributed assembly poses, and the advantages that it can lead to when implemented correctly. This simulation is designed so that we may expand upon it once a broader understanding of the concepts related to distributed assembly are achieved.

### 3.1 One Robot Building a 2-D Structure

This simulation involves one mobile robot, a desired structure, and a block cache location. Let $\mathcal{W}$ denote the workspace, and let $\mathcal{S}$ be the shapemap, which is a map of the target structure. Included in $\mathcal{S}$ are the positions of the block cache, $C$, as well as the starting point of the structure $\mathbf{m}$, or marker. We assume $\mathbf{m}$ is already placed, and the locations of both $\mathbf{m}$ and $C$ are known by the robot. We also assume that the robot has the ability to locate $\mathbf{m}$ and $C$, know where its current position is in $\mathcal{W}$, knows $\mathcal{S}$, and can avoid obstacles like the partially built structure. The robot is also capable of following the perimeter of the structure in the clockwise direction and recognizing when it has reached a legal attachment site 1 .

In order to get from one position to another, for example, from $C$ to the structure, the robot uses an algorithm called $A^{*}$ to define the path. This algorithm is expanded upon in Appendix B. $A^{*}$ is a graph search method which provides a route from the robot's current position in $\mathcal{W}$ to the desired final position in $\mathcal{W}$, taking into account any obstacles in the workspace.

### 3.1.1 Running the Simulation

The simulation begins with the robot placed at a starting point $p_{s}$ defined by the user. We first instruct the robot to drive to $C$ to retrieve the building material. Once the robot has the block, it drives to $\mathbf{m}$. The robot always drives from $C$ to $\mathbf{m}$ because $\mathbf{m}$ is the only position in the target structure whose position is known by the robot. For this navigation method to work, $\mathbf{m}$ must lie on the perimeter of the target structure. If $\mathbf{m}$ is not on the perimeter, then the robot must use more intuition once it has reached the structure. Since $\mathbf{m}$ is an occupied space, driving to $\mathbf{m}$ means the robot will drive to a spot next to it and orient itself such that it can begin to follow the perimeter of the structure in a clockwise manner.

To assemble the structure, we use a method similar to the one seen in [21, 22]. The robot drives around the structure until it arrives at $\mathbf{l}$. The location of $\mathbf{l}$ is either the beginning of a row, or the spot next to the last placed block in the partially built row. The robot will start building a new row when it recognizes that it has reached the beginning of a row as long as there are no other open rows. It can figure this out based on its knowledge of $\mathcal{S}$. The simulation is complete when all of the blocks in the target structure have been placed.

### 3.2 Two Robots Building a 2-D Structure

After several successful attempts of creating differently shaped structures using one robot, we now expand the simulation to include another robot. In this expanded scenario, the robots still work independently and do not assemble the target structure simultaneously. This is because both the controller and the assembly strategy are centralized. This leads to the solution being no more efficient as the single-robot example. Figure 3.1 depicts the two-robot assembly simulation in four steps.

This example provides an introduction to general robotic assembly, and begins to incorporate a distributed aspect by adding another robot. Already, we begin to see how challenging this problem can be. In order to have a true distributed system, we must define more rules for both the robots and the assembly strategy. By distributing the system such that each robot is controlled independently and different portions of the structure can be tended to simultaneously, we will achieve a much more efficient system. To tackle the problem of distributing the workload, we devise a new assembly strategy.


Figure 3.1: The robots are represented by the green dots with arrows marking their orientations. The black square around the robot indicates that the robot is carrying a block. The red square represents $\mathbf{m}$ and the blue squares represent $C_{i}$ where the blocks are obtained. The yellow squares represent $\mathcal{S}$, and the black squares around the yellow ones represent placed blocks. This figure shows four screen shots of the simulation as it is happening.

## Chapter 4

## Distributed Assembly in 2-D

By limiting the legal attachment sites to only the current row being assembled, as done in the previous experiments, we reduce the possibilities for sites where the robot is allowed to place a block. A shapemap may be very large and may expand in different directions, so this task could potentially take a very long time given an inefficient assembly strategy. Also, the idea of intelligent building materials [22, 23], which was not utilized in the previous experiment, is explored here.

### 4.1 Methodology

### 4.1.1 Scoring Cells Adjacent to the Structure

To take advantage of what the structure is capable of doing, i.e. communicating with the robot, we define a new method for assembly. For this method, we explore the concept of assigning a score to each location in $\mathcal{S}$. The scoring strategy is based on how many adjacent cells to point $p_{i}$ are occupied, and are therefore a part of the already built structure. Cell $p_{j}$ is considered adjacent to $p_{i}$ if $p_{j}$ is in $p_{i}$ 's neighborhood of 4 or neighborhood of 8 , depending on the experiment. These neighborhoods are
composed of the 4 or 8 cells surrounding $p_{i}$, as seen in Figure 4.1. Therefore, a $p_{i}$ is given a score based on how many occupied cells' 4- or 8-neighborhoods $p_{i}$ lies in. Every time a new block is placed, each $p_{i}$ is re-scored to take the new placement into account. We define $\mathbf{s}$ to be the list of scores for each cell, and $s_{i}$ to be the score of the corresponding cell $p_{i}$.


Figure 4.1: Neighborhood of 4, seen in (a), and neighborhood of 8, seen in (b)

### 4.1.2 Different Assembly Techniques

Using this scoring idea, a new experiment was created. This experiment differs from the previous one in that no attempt was made to simulate the robot driving to and from the structure. For this simulation, we assume that the structure is planar and has no internal holes. We also assume that the robots can query the blocks, and that the blocks have limited processing capabilities, allowing them to store local data. The blocks can then communicate this data to the robot when asked for it. This allows the robot to learn its exact location in $\mathcal{W}$ when it arrives to the structure, and know where it is at all times while it is circling the structure.

In this simulation, a random cell along the perimeter of the already built structure is denoted as $h$. One of $h$ 's out-facing sides is chosen at random as the side at which the robot arrives. This allows us to assume that the robot is capable of picking up a block and delivering it to the structure without having to simulate it. A depiction
of the simulation may be found in Figure 4.2. This method also does not require the marker to be on the perimeter of the structure. Despite not actually simulating the robot's motions graphically, the distance that the robot travels around the structure from $h$ to the attachment site is still recorded. Like the previous simulation, the robot is directed to travel in a clockwise manner once it has reached the structure until it arrives at the attachment site. When the robot arrives at the structure, it calculates the next block placement site, $b$, based on its knowledge of the state of the unfinished structure, and one of several block placement strategies. When $b$ has been determined, the robot proceeds to that spot.


Figure 4.2: Depiction of the scoring simulation: The empty squares represent $\mathcal{S}$. The symbols on the blocks adjacent to the structure indicate the score for those spots. The individual scores can be seen in the figure. The arrival site $h$ is depicted by a green square, and the side that the robot arrives at is indicated by the triangle.

Three different methods were developed to assemble the structure using the scoring technique. For each method, the robot was tasked to assemble four structures of different shapes and sizes. The starting point for each shape was the same point near
the middle of the structure, and was the same for each assembly method.

## Block Placement Based on Highest Score

Now that we have the scoring procedure clearly defined, different assembly schemes can be defined. The first of these is simply to place $b$ in the spot with the highest score. The location of $b$ is determined by Formula 4.1.

$$
\begin{equation*}
b=\underset{i}{\arg \max } s_{i} \tag{4.1}
\end{equation*}
$$

Since there will inevitably be instances where the highest score occurs in multiple locations, the tiebreaker is simply the one that occurs first when the algorithm is searching through $\mathcal{S}$ to find all of the scored sites.

By placing blocks in the spaces with the highest score, holes in the structure can be avoided. The advantage to this assembly strategy is that the construction is done in an orderly manner. This is achieved because as soon as a gap is created, it is filled in due to the high scores the empty neighboring cells will receive. Pseudocode for this method is found in Algorithm 4.1.

```
Algorithm 4.1 Block Placement Based on Highest Score
    while structure not complete do
        score map
    3: sort \(\mathbf{s}\) from highest to lowest score
        generate \(h\)
        place block in top cell on list
        log distance traveled
    end while
```

This method is not efficient, because the robot is still ordered to travel around the structure in only one direction, so $b$ may still be very far from $h$. This observation leads to the next method, which is to take the distances of every $p_{i}$ into account, as well as the score.

## Block Placement Based on Score and Distance from Robot

The second method aims to shorten the travel time of the robot while still taking the scoring method into account. To do this, we calculate a new number. This number is found by subtracting every $s_{i}$ from the distance $d_{i}$, where $d_{i}$ is the number of steps around the perimeter it takes to get from $h$ to $p_{i}$ in the clockwise direction, and $\mathbf{d}$ is a vector that contains every $d_{i}$. To find $b$, we use Equation 4.2.

$$
\begin{equation*}
b=\underset{i}{\arg \max }\left(k_{1} s_{i}-k_{2} d_{i}\right) \tag{4.2}
\end{equation*}
$$

This simple formula yields a highly scored legal attachment site that is close to $h$. If there is no $p_{i}$ with a high score near $h$, then the robot will simply attach the block at a lower scored site closer to $h$. The weighting constants $k_{1}$ and $k_{2}$ are used to give either the distance or the score a higher importance than the other. For this experiment, $k_{1}=k_{2}=1$ was used. The algorithm used for this assembly strategy can be seen in Algorithm 4.2

```
Algorithm 4.2 Block Placement Based on Score and Distance from \(h\)
    while structure not complete do
        calculate \(d_{i}\) for each \(p_{i}\)
        calculate \(b\) using Equation 4.2
        place block at \(b\)
        log distance traveled
    end while
```

Interestingly, this method proved to be less efficient than the first one. Because the robot would often place blocks close to $h$, and given the fact that $h$ is chosen randomly, the assembly turned out to be less orderly than the initial method. This leads to the third method, which aims to maintain the orderliness of the highest-score method, but shorten the distance traveled wherever possible.

## Block Placement Based on Control of Perimeter Size

The third method combines the ideas of the previous two methods. The first step of this method is the same as the first method. The map is scored and the highest scoring cells are chosen as the next possible attachment sites. Let $r_{i}$ be the perimeter of the structure that would be created if $b$ is chosen to be $p_{i}$, and let $R$ denote the set containing all $r$ values.

Once all values of $s_{i}$ are calculated, we determine which of these sites would create the smallest perimeter if a block should be placed there. If there is one spot where this is the case, then the robot will deliver the block to this site. If there are multiple spots that have the highest score, they are added to the set $S_{\text {max }}$. If there are multiple agents in $S_{\text {max }}$ that create the same minimum perimeter $r_{\text {min }}$, they are added to $R_{s, m i n}$. These variables are defined in Equations 4.3 and 4.4. The robot will deliver the block to the closest $p_{i}$ in $R_{s, \text { min }}$. Equation 4.5 defines the calculation of $b$ for this method. The algorithm that is used to simulate this method is outlined in Algorithm 4.3. This method ensures that the structure is being completed in a responsible, organized manner, yet still attempts to find the shortest distance possible.

$$
\begin{gather*}
S_{\max }=\left\{s_{i} \mid s_{i}=\max (\mathbf{s})\right\}  \tag{4.3}\\
R_{s, \min }=\left\{\underset{i}{\arg \min }\left\{r_{i} \mid s_{i} \in S_{\max }\right\}\right\}  \tag{4.4}\\
b=\underset{i}{\arg \min }\left\{d_{i} \mid r_{i} \in R_{\min }\right\} \tag{4.5}
\end{gather*}
$$

```
Algorithm 4.3 Block Placement Based on Perimeter Control
    while structure not complete do
        calculate s
    3: sort \(\mathbf{s}\) from highest to lowest score
        for \(i=1\) to length of \(S_{\max }\) do
            \(r_{i}=\) perimeter of structure that placing block at this point would create
6: end for
        if there is only one \(p_{i}\) corresponding to the smallest perimeter then
            \(b\) is \(p_{i}\) corresponding to the minimum value in
        else
            \(b\) is the minimum \(p_{i}\) value closest to \(h\)
        end if
    end while
```


### 4.2 Experimental Analysis

An experiment was designed to implement the shortest distance assembly strategy and the perimeter assembly strategy. These experiments involved one $\mathrm{M}^{3}$ building a 2-D 5x5-block square. Figure 4.3 shows the experimental setup with the structure's position and the positions of the two block caches. The assembly sequence for each strategy was determined a priori by the Matlab simulations described above, with user defined $h$ locations rather than random $h$ locations. These points were picked based on which cache the robot would be coming from for each block placement.


Figure 4.3: Experimental setup for scoring experiment

To precisely navigate the workspace, the robot used the $A^{*}$ algorithm to compute
waypoints every 250 mm to drive back and forth between the structure and the block caches. This was done to discretize the robot's motion so that it could precisely follow the perimeter of the structure during the assembly process. To maintain consistency with the simulation, the robot was also instructed to only travel in the clockwise direction around the structure. The structure was assembled having the robot drive along the trajectories from the block cache to the structure. When the robot arrived at the spot on the grid where a block needed to be placed, a manual placement was carried out by hand to simulate assembly. Figure 4.4 shows the experiment taking place and the final completed structure.


Figure 4.4: Photos of experiment taking place.

### 4.3 Results

### 4.3.1 Simulations

For each of the three assembly methods, 50 iterations were carried out for four different shapemaps. The four shapemaps can be seen in Figure 4.5. Distance data was recorded for all 50 iterations for each assembly method and for each map. The data may be found in Table 4.1. The data in the table includes an average of the total distance traveled over 50 iterations, the standard deviation of this data, and the maximum and minimum distance traveled in the set of 50 runs for each assembly
method. Furthermore, the plots in Figure 4.6 contain data for the average distance traveled for each block placement for all assembly methods and shape-maps.


Figure 4.5: The four shape-maps evaluated in this simulation

### 4.3.2 Experiments

For each assembly strategy, three trials were carried out. Only one shapemap was used in experimentation. The distances for each trial were recorded, and the data is contained in Table 4.2.

| Assembly Method | Average Total <br> Dist. Traveled | Standard Dev. of <br> Dist. Traveled | Minimum <br> (measured in steps) Traveled | Maximum <br> Dist. Traveled |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | Small Square |  |  |  |  |  |  |
| High Score | 1627.28 | 94.71 | 1362 | 1868 |  |  |  |
| Distance | 1708.10 | 102.69 | 1519 | 1938 |  |  |  |
| Perimeter Control | 1493.48 | 104.36 | 1228 | 1651 |  |  |  |
| High Score | 11830.76 | Large Square |  |  |  |  | 12441 |
| Distance | 13315.24 | 403.27 | 10944 | 14132 |  |  |  |
| Perimeter Control | 10515.24 | 381.55 | 12405 | 11878 |  |  |  |
| High Score | 1465.78 | 512.84 | 9636 | 1729 |  |  |  |
| Distance | 1460.00 | 95.07 | Small Maze | 1793 |  |  |  |
| Perimeter Control | 1559.72 | 122.83 | 1252 | 1788 |  |  |  |
| High Score | 21278.18 | 97.57 | 1199 | 1307 | 22665 |  |  |
| Distance | 22487.56 | 624.68 | Large Maze | 24488 |  |  |  |
| Perimeter Control | 21198.88 | 867.41 | 19949 | 22916 |  |  |  |

Table 4.1: Results for distance traveled by robot for each assembly method given four different shape-maps.

| Assembly Method | Trial 1 | Trial 2 | Trial 3 | Average |
| :---: | :---: | :---: | :---: | :---: |
|  | (distance in meters) |  |  |  |
| Distance | 291.03 | 296.00 | 303.85 | 296.96 |
| Perimeter Control | 325.48 | 323.42 | 324.26 | 324.39 |

Table 4.2: Results for distance traveled by robot for each assembly method. The robot traveled an average of $9.2 \%$ farther for the perimeter control method.

### 4.4 Discussion

Now that we have accumulated a set of data, we can begin to analyze the results. Looking at numbers for the two square maps in Table 4.1 lead to some interesting conclusions. The attempt to shorten the distance traveled by taking the distance into account in the block placement function failed. The reason for this becomes clear when we study Figures $4.5(\mathrm{a})$ and $4.5(\mathrm{~b})$. Ignoring the perimeter method for the time being, we see that the robot is traveling a shorter distance during the time when the structure is roughly $20 \%$ complete until it is roughly $80 \%$ complete. It turns out that because the robot is trying to place blocks closer to $h$, it is creating a more complex structural shape in the process. This causes the robot to travel farther distances in the middle of the assembly than the original high score method. When
the structure is near completion, though, the numbers begin to even out. This is because the shape of the structure is near its desired state, and thus similar for all cases.

The results from the first two assembly strategies led to the idea for the third. If creating a more complex perimeter caused the robot to travel larger distances, then controlling the shape of the perimeter should shorten it. For the small square, large square, and large maze, this proves to be the case. For these three maps, the perimeter control method provides the best results. Interestingly, for the small maze, the distance method proves to be the most efficient. The reason for this is the low number of attachment locations during a given assembly of this map. At any given point during the assembly, the robot has far fewer options for legal block placement sites than the other three maps. Also, for the small maze, the perimeter is defined mainly by $\mathcal{S}$ rather than where the robot is placing blocks. Even though this may be a special case, it shows that each assembly method has upsides and downsides, and each has a scenario where it can be relevant.

The experimental results prove this point. As noted in Table 4.2, the distance method proves to supply the faster assembly trajectories, with the robot traveling an average of 27.43 meters less each time. Because the structure never gets big enough for complexities to arise, traveling the shortest distance for each block placement is better for this circumstance.

Unlike the example from Chapter 3, where only one legal attachment site exists for any given time step $t$, multiple legal attachment sites exist in these new examples. This leads to the opportunity to introduce more robots during the assembly process. Although these experiments only involve one robot, a new set of rules could be defined to utilize more robots. Using a team of robots in this scenario would lead to faster assembly times. We will explore multi-robot techniques in the next chapter.


Figure 4.6: Plot of the average distance traveled for each block placement over 50 iterations of each assembly method

## Chapter 5

## Decentralizing the Assembly Tasks

Thus far, the simulations we created have dealt mainly with single robot systems. Even the simulations expanded to include multiple robots have been centralized and have not taken advantage of having more than one robot assembling the structure. The next step is to develop a method for distributing the assembly tasks to all of the robots in our system so that they can assemble the different parts of the structure simultaneously. This will speed up the assembly time, but it will also introduce new challenges. We first examine ways to divide the structure so that different pieces can be allocated to different robots.

### 5.1 Equal Mass Decomposition of Target Structures Using Voronoi Diagrams

To decompose structures, we begin with the idea of using a Voronoi diagram to split structures based on equal mass partitions. Voronoi diagrams create a tessellation of the workspace, and they can be used to create a scenario such that every robot has the same amount of work to do. An in-depth explanation of how these diagrams are
realized may be found in Appendix C.
To achieve equal-mass partitions, we first assign a uniform mass to every position in $\mathcal{S}$ and decompose this map using a Voronoi diagram. Then, random positions on the map are selected to be the Voronoi points $\left\{\mathbf{p}_{1}, \ldots, \mathbf{p}_{n}\right\}$, where $n$ is the number of robots that will be building the target structure. Robot $i$ will be responsible for constructing the portion of the structure within $C_{V}^{i}$, which is calculated in Equation C.1. Once placed, each $\mathbf{p}_{i}$ moves to a position such that every $C_{V}^{i}$ contains roughly the same mass $M_{C_{i}}$. This is achieved by designing a cost function $\mathcal{H}$, which is defined as

$$
\begin{equation*}
\mathcal{H}=\mathcal{H}_{0}-\prod_{i=1}^{n} M_{C_{i}} \tag{5.1}
\end{equation*}
$$

where

$$
\begin{equation*}
M_{C_{i}}=\int_{C_{V}^{i}} \phi_{t}(\mathbf{q}) d \mathbf{q} \tag{5.2}
\end{equation*}
$$

and

$$
\begin{equation*}
\mathcal{H}_{0}=\left(\frac{1}{n} \sum_{i=1}^{n} M_{C_{i}}\right)^{n}=\left(\frac{1}{n} \int_{\mathcal{W}} \phi_{t}(\mathbf{q}) d \mathbf{q}\right)^{n} \tag{5.3}
\end{equation*}
$$

Here, $\mathbf{q}_{i}$ is a point in $\mathcal{W}$ and $\phi_{t}$ is the target density function.
The desired mass in each $C_{V}^{i}$, therefore is the arithmetic mean $\overline{M_{C}}$ defined by

$$
\begin{equation*}
\bar{M}_{C}=\frac{1}{n} \sum_{i=1}^{n} M_{C_{i}} . \tag{5.4}
\end{equation*}
$$

We see from this set of equations that as each $M_{C_{i}}$ approaches $\bar{M}_{C}$, the cost function $\mathcal{H}$ will approach zero. It is with this knowledge that we aim to minimize $\mathcal{H}$ and achieve roughly equal mass in each $C_{V}^{i}$ [5].

One advantage to this method is that it can be done offline, before the assembly actually begins. Once the robots begin to assemble the structure, they only have to work within their pre-partitioned space. A simulation was carried out using a simple square structure and a team of 6 homogeneous robots. We can see the results of the partitioning in Figure 5.1. Notice how in Figure 5.1(a), the variance in size of the partition is quite large, while in Figure $5.1(\mathrm{~b}), \mathcal{H}$ has been minimized to the point where the variance between each $V_{i}$ is much smaller.


Figure 5.1: Voronoi decomposition of a simple square structure with uniform mass. The shaded region represents the structure, the black dots represent the Voronoi points and the lines represent the edges.

We quickly realize that this method, although dividing the mass equally amongst the robots, does not create an ideal section for each robot because the geometry of the structure is not taken into account. Figure 5.2 shows four more structures divided into six equal-mass partitions. The resulting numbers for each of these structures, as well as the square in Figure 5.1 can be seen in Table 5.1. Again, we see that each map has roughly the same mass in $C_{V}^{i}$, but the decomposition is far from ideal. Looking at Figures $5.2(\mathrm{c})$ and $5.2(\mathrm{~d})$, it is easy to see that as the structure is assembled, the number of inter-agent collisions is sure to increase compared to the structure in Figure 5.1. Also, in the cases of Figures $5.2(\mathrm{a})$ and $5.2(\mathrm{~d})$, the result might even be
unrealizable unless the robot traps itself in the middle of the structure. In order to achieve the desired level of parallelization, we need another approach. What we want to achieve is a navigation and geometry aware task partitioning strategy.


Figure 5.2: Four maps with their final equal-mass Voronoi cells.

| Shape-map | \% of Total Mass in Each Cell |  |  |  |  |  | Std Dev |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 1 | 2 | 3 | 4 | 5 | 6 |  |
| Square | 18.4 | 15.7 | 14.9 | 16.8 | 17.9 | 16.2 | 1.3 |
| B | 12.8 | 17.1 | 18.1 | 16.9 | 19.2 | 16.0 | 2.2 |
| M | 16.1 | 17.7 | 13.4 | 17.4 | 17.7 | 17.7 | 1.7 |
| Maze | 14.1 | 16.4 | 19.0 | 16.7 | 16.5 | 17.3 | 1.6 |
| Window | 14.7 | 16.1 | 15.9 | 16.6 | 18.9 | 17.8 | 1.5 |

Table 5.1: Mass distribution across the Voronoi partitions for the structures shown in Figures 5.1(b) and 5.2.

### 5.2 Maintaining Navigation Awareness

### 5.2.1 Weighted Mass-Based Partitioning

To improve upon the simple mass based partitioning, we examine ways to take the shape of the target structure into account. Using the same cost function $\mathcal{H}$, we can still try to do this. In the simple mass-based partitioning method, every $\mathbf{q}_{i} \in \mathcal{S}$ has equal mass. In an attempt to make certain parts of the structure more important, we can add weighting schemes to the maps. These weighting schemes attempt to
highlight the importance of narrow entrances and passageways within the structure because they introduce navigational challenges as the structure is being built.

Figure 5.3 shows a B shaped map with four different weighting schemes. It is clear that the attempt to improve the partition in terms of navigability fails. The three different weighting schemes all provide almost identical results to the original partition. A new approach is necessary if we want to achieve a more navigation-aware decomposition of the map.


Figure 5.3: Different weighting schemes applied to the B map.

### 5.2.2 Defining an Optimization Strategy for Navigation-Aware Partitions

In our workspace $\mathcal{W}$, we define the free space as $\mathcal{W}_{f}$, where $\mathcal{W}_{f}=\mathcal{W} / \mathcal{S}$. We define a roadmap graph of important navigational points as an undirected graph $G_{R}=$ $\left(V_{R}, E_{R}\right)$, where $V_{R}$ is the set of vertices $\left\{v_{R}^{1}, \ldots, v_{R}^{j}\right\}$ and $E_{R}$ is the set of edges $\left\{e_{R}^{1}, \ldots, e_{R}^{k}\right\}$. For a team of $n$ mobile robots, the objective is to partition $\mathcal{S}$ into $T_{i}$ connected subsets such that $\mathcal{S}=\cup_{i=1}^{n} T_{i}$ and $\left|\phi\left(T_{i}\right)-\phi\left(T_{j}\right)\right|$ is minimized for every $\{i, j\}$ pair. Here, $T_{i}$ denotes the subset of $\mathcal{S}$ assigned to robot $i$ for assembly.

We will now see that the same method can be used within the structure to find other important node locations. To define these new locations, we assume a convex cell decomposition of $\mathcal{S}$ and construct an undirected graph $G_{\mathcal{S}}=\left(V_{S}, E_{S}\right)$ to represent the cells within the desired structure. Each convex cell in $\mathcal{S}$ is represented by a vertex
in $V_{S}=\left\{v_{S}^{1}, \ldots, v_{S}^{l}\right\}$ and the edges in $E_{S}$ represent adjacency relationships between the vertices in $V_{S}$. We refer to the graph $G_{\mathcal{S}}$ as the structure graph.

We next define the adjacency matrix $A_{S}$ for the structure graph $G_{\mathcal{S}}$ is given by

$$
A_{S}=\left[a_{S_{i} j}\right]= \begin{cases}1 & v_{i}, v_{j}, \in V_{S} \text { and } v_{i}, v_{j} \text { share an edge }  \tag{5.5}\\ 0 & \text { otherwise }\end{cases}
$$

Given the roadmap and structure graphs, $G_{R}$ and $G_{S}$, the objective is to obtain a navigation-aware partitioning of $\mathcal{S}$ into $T_{i}$ to enable parallel assembly of $\mathcal{S}$ by a team of $n$ homogeneous robots. In other words, for every $v_{S}^{i} \in V_{S}$, the goal is to assign every $v_{S}^{j} \in V_{S}$ to a $T_{i}$ such that for every $v_{S}^{p}, v_{S}^{q} \in T_{i}, a_{S_{p} q}=1$ and there exists a $v_{S}^{r} \in T_{i}$ such that a feasible path for every point $\mathbf{q} \in v_{S}^{r}$ to a vertex in $V_{R}$ exists, and denote this as $v_{S}^{r} v_{R}^{t}$. To achieve this, we define the following optimization problem as the minimization of:

$$
\begin{equation*}
\sum_{i \in\{1, \ldots, n\}, i \neq j}\left(\phi\left(T_{i}\right)-\phi\left(T_{j}\right)\right)^{2} \quad \text { s.t. } \sum_{i=1}^{n} T_{i}=\phi(\mathcal{S}) \tag{5.6}
\end{equation*}
$$

This equation is similar to Equation 5.1, with the main difference being that we are grouping subsets of $T$ rather than individual spots $\mathbf{q}$.

In general, solutions to Equation 5.6 may be difficult to obtain since multiple solutions may exist with no guarantees on global optimality. For small enough $n$ and reasonably sized roadmaps and structure maps, an exhaustive search can be used to determine the optimal solution. However, different from existing approaches like Equation 5.1, the solution to Equation 5.6 guarantees a feasible path within the workspace exists for every element in $T_{i}$ for a given assembly strategy.

### 5.2.3 Using the Brushfire Algorithm to Partition Target Structures

In an attempt to apply the optimization strategy in Equation 5.6 we will utilize the Brushfire algorithm. This algorithm is described in Appendix D. The algorithm results in the highest values being at points that are farthest from both the structure and the edge of the workspace. It turns out that the ends and intersections of these lines can be useful in our navigation strategy around the structure. In other words, we will use these points as $V_{R}$.

To demonstrate this, a contour plot of 6 different structures with the brushfire algorithm applied to the outside of the structures can be seen in Figure 5.4. The contour plot simply assigns a color depending on the number for that spot on the grid in Figure D.1, therefore, positions with the same number will be the same color. This will lead to an image whose elevations are easy to distinguish. For example, in Figure 5.4(a), the dark blue represents the lower numbers, while the yellow and red represent higher numbers. In these plots, the points of interest become apparent upon observation.

(a)

(b)

(c)

(d)

(e)

(f)

Figure 5.4: Six shapemaps with the brushfire algorithm applied to the outer edge of each structure and map border.

The usefulness of the brushfire algorithm does not stop at the navigation level. Using the set of $V_{R}$ derived from the plots in Figure 5.4, we set up $G_{S}$ such that we ensure that every $v_{S}^{i}$ is reachable from at least one $V_{R}$ once the structure has been partitioned and during construction, as defined in Section 5.2.2. We refer to

Figure 5.5 to find the locations of each $v_{S}^{i}$. Like Figure 5.4, Figure 5.5 produces plots with recognizable nodes. These plots are easy to create, but they still lead to some ambiguity. Figure 5.6 shows each structure with $G_{R}$ and $G_{S}$ clearly defined.

(a)

(b)

(c)

(d)

(e)

(f)

Figure 5.5: Six shapemaps with the brushfire algorithm applied to the inside of each structure.

(a)

(b)

(c)

(d)

(e)

(f)

Figure 5.6: Six shapemaps with $V_{R}$ shown with blue and red dots and $V_{S}$ shown with yellow dots.

We can now begin to think about partitioning the structure using these ideas. The first idea is to use the values residing in $V_{S}$ as Voronoi points. This results in a more organized decomposition than before, but we still don't have a very "navigationaware" partitioning. Figure 5.7 shows the decomposition of four structures using this method. Figure 5.7 (c) is an especially good example of a poor navigation-aware decomposition.


Figure 5.7: Navigation-aware partitioning of structures using $V_{S}$ as Voronoi points.

Based on the poor results of using $V_{S}$ to determine the center of the cells, we will instead try to use them as the boundaries for the cells. Figure 5.8 shows the new decomposition using this approach. These maps provide very good navigation-aware partitioning. As opposed to the Voronoi results, these maps provide a more logical decomposition. Applying the proposed strategy given in Equation 5.6, we evaluate the target structures with convex cell decompositions in Figure 5.8. For a team of 6 robots, Figure 5.9 shows the resulting navigation-aware partitions assigned to each robot. Table 5.2 shows the percentage of the total mass contained in each $T_{i}$ for each target structure shown in Figure 5.9. As expected, the number of elements to be assembled within each $T_{i}$ is not the same for every member of the six robot team.


Figure 5.8: Decomposition using $s$ nodes as boundaries for cells.


Figure 5.9: Navigation-aware task partitioning where the boundaries for $T_{i}$ are denoted by the thicker lines.

### 5.3 Defining Structural Complexity

We wish to define another cost function that takes the complexity of a structure into account. The hope is that by defining a complexity based cost function, we will be able to partition the target structure in such a way that each robot has a similar amount of work, yet also results in an easily navigable workspace.

In general, given a two-dimensional structure, we have developed an intuitive understanding of the complexity of the structure to be assembled. For example, it is easy to understand how a target structure like the "smiling monster" seen in Figure 5.9(e) would be defined as "more complex" than the square structure in Figure 5.1. Still, the term "complexity" is subjective, and there will be many cases where the type of robot, or the building materials being used, will add to a given structure's complexity without it being initially apparent. Since we cannot account for every possible robot or building material, we will define a metric that can provide an estimate to enable

| $\mathcal{S}$ | \% of Total Mass in Each Partition |  |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 1 | 2 | 3 | 4 | 4 |  |  |
|  | $\left.(\%) \phi_{j}\right)^{2}$ |  |  |  |  |  |  |
| A | 11.6 | 15.8 | 24.3 | 15.8 | 11.6 | 20.9 | 7.7 |
| B | 13.1 | 11.1 | 15.1 | 23.4 | 14.3 | 23.1 | 8.3 |
| M | 19.5 | 20.7 | 20.0 | 20.7 | 9.5 | 19.5 | 9.2 |
| Maze | 21.5 | 13.8 | 16.3 | 17.7 | 13.9 | 16.8 | 2.4 |
| Smiling Monster | 22.0 | 20.1 | 12.9 | 16.6 | 14.2 | 14.3 | 4.0 |
| Window | 18.9 | 15.8 | 13.0 | 18.6 | 14.9 | 18.8 | 1.8 |

Table 5.2: Mass distribution across the navigation-aware partitions $T_{i}$ for the structures shown in Figure 5.9 where $\phi_{i}=\phi\left(T_{i}\right)$. The boundaries for $T_{i}$ are denoted by the thicker lines.
comparisons, and inform the design of distributed strategies.

### 5.3.1 Problem Definition

To define complexity, we will again use $\mathcal{S}$ to represent our target structure. We will consider the workspace $\mathcal{W}$ to be a convex polygonal space and obstacle free, until the assembly of $\mathcal{S}$ begins. Let $\mathbf{q}$ denote a point in $\mathcal{W}$, and as before, let $\phi_{t}(\mathbf{q})$ be the target mass density function for $\mathcal{S}$. Let $S_{i}$ be the $i^{\text {th }}$ connected subset of $\mathcal{S}$, and $\mathcal{M}$ be the number of disconnected members of $\mathcal{S}$ such that $\mathcal{S}=\cup_{i=1}^{\mathcal{M}} S_{i}$. Additionally, we denote the $j$ massless connected subset enclosed in $S_{i}$ as $O_{i j}$, i.e. $\phi_{t}(\mathbf{q})=0$ for $\mathbf{q} \in O_{i j}$, and define $O_{i}=\cup_{j=1}^{m_{i}} O_{i j}$ where $\mathcal{O}=\cup_{i=1}^{M} O_{i}$. Next, we let $E$ and $H$ denote the number of entrances into and hallways in $\mathcal{S}$. We will consider any opening into the structure as an entrance.

Next, we let $\partial \mathcal{S}$ and $\partial \mathcal{O}$ denote the boundaries of the structure, and of the set of massless connected components respectively. We denote the length of these boundaries by $L($.$) . The area of each of these sets will be given by A($.$) , and let w_{i}, v_{i}$, and $l_{i}$ represent the width of entrance $i$, the width of hallway $i$, and the length of hallway $i$ respectively. Since the desired metric must be independent of the size of individual robots, we define $R$ as the radius of the smallest circle that circumscribes the robot.

To understand all of these variables, we look at a couple of examples. For the target structures in Figure 5.8, the yellow shaded areas represent $\mathcal{S}$. For Figure $5.8(\mathrm{c}), M=1$ since there is only one connected component of $\mathcal{S}$. Further, $\mathcal{O}=0$ and $E=H=1$. Figure $5.8(\mathrm{~d})$ shows a structure where $M=2, \mathcal{O}=0, E=2$ and $H=1$. Finally, Figure 5.8(e) is an example with $M=1$ and $O_{i}=\cup_{j=1}^{2} O_{i j}$ with each $O_{i j}$ given by the interior empty squares, and $E=H=1$.

With all of the variables laid out and understood, we can now define the complexity of a given $\mathcal{S}$ using Equation 5.11.

$$
\begin{gather*}
c_{1}=k_{1}\left(\frac{L(\partial \mathcal{S})}{\sqrt{A(\mathcal{S})}}\right)^{M}  \tag{5.7}\\
c_{2}= \begin{cases}k_{2}\left(\frac{L(\partial \mathcal{O})}{\sqrt{A(\mathcal{O})}}\right)^{\sum_{i=1}^{M} m_{i}} & \text { if } \mathcal{O}>0 \\
0 & \text { otherwise }\end{cases}  \tag{5.8}\\
c_{3}= \begin{cases}k_{3}\left(\sum_{i=1}^{E} \frac{2 R}{w_{i}}\right)^{E} & \text { if } E>0 \\
0 & \text { otherwise }\end{cases}  \tag{5.9}\\
c_{4}= \begin{cases}k_{4}\left(\sum_{i=1}^{H} \frac{l_{i}}{v_{i}}\right)^{H} & \text { if } H>0 \\
0 & \text { otherwise }\end{cases}  \tag{5.10}\\
C=c_{1}+c_{2}+c_{3}+c_{4} \tag{5.11}
\end{gather*}
$$

where $k_{1}, k_{2}, k_{3}$ and $k_{4}$ are weighting positive constants.
Equation 5.7 captures the geometric complexity of the target structure $\mathcal{S}$. In general, given $\mathcal{S}$, a larger discrepancy between the perimeter of the structure and the normalized area, i.e. $\sqrt{A(.)}$, implies more protrusions and indentations associated with the structure. In addition, as $M$ increases, the workspace becomes more complex as $\mathcal{S}$ is being assembled, since every $S_{i}$ introduces an obstacle into the environment.

Similarly, the Equation 5.8 describes the complexity introduced by massless regions enclosed in $S_{i}$ for all $i=1, \ldots, M$. Holes within a structure pose significant challenges in the synthesis of provably correct distributed algorithms unless an assembly prioritization scheme can be imposed for certain portions of $\mathcal{S}$.

Finally, Equations 5.9 and 5.10 describe the complexity introduced by the presence of narrow entry points into and hallways within $\mathcal{S}$. Such features can introduce significant traffic congestion for any robot trying to operate in the space. As the widths of the entrances and hallways shrink, the complexity increases since localization and navigation errors can detrimentally affect the performance of the entire system.

Although Equation 5.11 as a whole represents the complexity of the distributed assembly task $C$, it is important to note that each individual term can also serve as an independent measure for a given structure or task. These measures can be used to adjust the target mass density function $\phi_{t}(\mathbf{q})$ to affect the partitioning of the assembly task or to be used to determine the order in which components must be assembled.

The complexity measures described here can now be applied to some example target structures. The values for each measure can be found in Table 5.3. These values were obtained by assuming $R=1$ and $k_{1}=k_{2}=k_{3}=k_{4}=1$. Based on our chosen metric (given by Equation 5.11), the most challenging structure is the Window, which can be seen in Figure 5.8(f). The complexity of the structure arises

| $\mathcal{S}$ | $M$ | $\frac{L(\partial \mathcal{S})}{\sqrt{A(\mathcal{S})}}$ | $\sum_{i=1}^{M} m_{i}$ | $\frac{L(\partial \mathcal{O})}{\sqrt{A(\mathcal{O})}}$ | $E$ | $\sum \frac{w_{i}}{2 R}$ | $H$ | $\sum \frac{l_{i}}{v_{i}}$ | $C$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Square | 1 | 4.0 | 0 | NA | 0 | NA | 0 | NA | 4.0 |
| B | 1 | 10.2 | 1 | NA | 2 | 10 | 4 | 20.2 | 110.2 |
| A | 1 | 6.9 | 2 | 4.3 | 1 | 73 | 1 | 0.5 | 98.0 |
| M | 1 | 10.0 | 3 | NA | 1 | 10 | 2 | 14.5 | 20.0 |
| Maze | 2 | 15.9 | 4 | NA | 2 | 10 | 1 | 70.5 | 352.3 |
| Window | 1 | 9.2 | 6 | 10.2 | 0 | NA | 0 | NA | 1.1 e 6 |

Table 5.3: Complexity measures for various target structures
from the set of holes $\mathcal{O}$ within the target structure. One surprising result is the relatively low complexity value of the M structure, found in Figure 5.8(c). This is partly because the proposed complexity metric does not take into account the size of the robot team that will be used to assemble the desired structure [8].

## Chapter 6

## Methods for Assembly of Partitioned

## Structures

Now that we have defined a method for obtaining navigation-aware decompositions of a target structure, we must now define an assembly strategy for each robot within its given cell. To do this, we refer to the methods outlined in [25]. This strategy is based on our experimental setup, which includes a 3-D structure, rather than the 2-D structures dealt with in the previous sections. As a result, more physical constraints exist during the assembly of the structure. A block on the lower portion of the structure that must be placed prior to a block on the higher part of the structure is known as a supporting block. When a supporting block and the block above it are tasked to be placed by different robots, the scenario is denoted as a split constraint.

The purpose of the algorithm is to minimize four criterion. These are the total time it takes the robots to complete the structure, $T_{C}$, the time difference between robot $A$ "s completion time and robot $B$ 's completion time, $T_{D}$, the time each robot spends waiting for the other, $T_{W}$, and the number of split constraints, $C_{S}$. Dijkstra's algorithm, which is another graph search method similar to $A^{*}$, is used to help determine which assembly components are placed by which robots.

### 6.1 Experimental Setup

Two robots were tasked to assemble the target structure simultaneously. The target structure for experimentation is found in Figure 6.1. To assemble the structure, two different assembly sequences were obtained based on the methods found in [25]. These two sequences may be seen in Figure 6.2. Sequence 1 has 2 split constraints, while sequence 2 has 14 split constraints. Conflicts of this nature would require that a robot wait until the supporting block is placed before it could place its block. Because of the simplicity of the structure in our experiment, however, no robot needed to wait during either sequence.


Figure 6.1: Image of the target structure for experiment.


Figure 6.2: Two different assembly sequences - blocks shown in blue are assembled by Robot 1 and blocks in yellow are assembled by Robot 2 .

To begin the experiment, Robot 1 and Robot 2 were placed on opposite ends of the structure. The structure's base was pre-assembled so that the robots had a landmark to navigate to, using their LRFs. Also, because of the self-aligning grooves on the blocks, it was easier for the robots to place blocks on a pre-assembled base rather than directly on the floor. A diagram of this setup is seen in Figure 6.3. Figure 6.4 shows a sequence depicting the robotic assembly of the structure. The robots were given the global position of the structure and the positions of their respective parts caches. The robots traversed the workspace using the closed loop navigation strategy defined in Appendix A. Finally, each robot was given their respective assembly plan.


Figure 6.3: Diagram of the experimental setup

### 6.2 Experimental Results

Five iterations of each assembly sequence were carried out. During each of these trials, the time required to complete the experiment for each robot was recorded, as well as the distance traveled by each robot. The time spent waiting to place a block was also recorded, but there was no point during the experiment where either robot was required to wait for the other. The time and distance values are found in Tables 6.1 and 6.2.


Figure 6.4: Two robots assembling the structure during sequence 1 experiment

|  | Seq | Run 1 (s) | Run 2 (s) | Run 3 (s) | Run 4 (s) | Run 5(s) | Avg (s) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Exp. Time | 1 | 1028 | 1037 | 1060 | 1021 | 1027 | 1035 |
|  | 2 | 1069 | 1065 | 1030 | 1109 | 1174 | 1089 |
| Time Difference | 1 | 11 | 2 | 144 | 100 | 12 | 54 |
|  | 2 | 97 | 117 | 96 | 160 | 142 | 122 |

Table 6.1: Experimental results for 2 assembly sequences of 3-D structure

|  | Seq | Run 1 (m) | Run 2 (m) | Run 3 (m) | Run 4 (m) | Run 5 (m) | Avg (m) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Tot. Dist. Trav. | 1 | 31.58 | 31.30 | 31.27 | 31.32 | 31.34 | 31.36 |
|  | 2 | 32.10 | 32.25 | 32.18 | 32.33 | 32.22 | 32.22 |

Table 6.2: Experimental results for 2 assembly sequences of 3-D structure

## Chapter 7

## Concluding Remarks

This work introduces methods for decentralized distributed assembly, using a team of homogeneous robots. Using these methods, we show that assembly of a target structure is possible using our distributed assembly multi-robot testbed. In the process of describing the methods and algorithms in this work, we have learned the challenges that distributed assembly poses and provided ways to deal with these challenges to come to a successful realization. As research in distributed assembly continues, and robotic capabilities become greater, the likelihood of real world implementation of a distributed assembly system grows.

### 7.1 Future Work

The work done in Chapter 4 could be expanded upon. There are other methods that could be conceived to improve the current ones presented here. Also, proving correctness of assembly would go a long way in making these assembly methods viable techniques to use.

Regarding the navigation aware partitioning, given that the method presented does not necessarily result in similarly sized partitions, an online auctioning system
for robots to give sections of their partitions to neighboring robots would be useful in balancing the mass in each partition, while maintaining navigation awareness.

Finally, the experimentation presented in this work displays the use of one or two $\mathrm{M}^{3}$ robots. In order to truly show the advantages of our navigation aware partitioning ideas compared to the equal mass partitioning ideas, more robots must be introduced during experimentation. An experiment designed to compare these two methods would be useful in showing that complexity based partitioning would be a more realistic partitioning strategy.

## Appendices

## Appendix A

## Differential Drive Robot Kinematics for Closed-Loop Navigation

To move around the workspace, we assign various waypoints for the robots to drive to. To get from waypoint to waypoint, we must define equations so that we can give the individual wheels commands to get the robots to their desired poses. Let Equations A. 1 and A. 2 be

$$
\begin{align*}
& v=\frac{r\left(\dot{\phi}_{R}+\dot{\phi}_{L}\right)}{2},  \tag{A.1}\\
& \omega=\frac{r\left(\dot{\phi}_{R}+\dot{\phi}_{L}\right)}{2 l}, \tag{A.2}
\end{align*}
$$

where $v$ is the robot's forward velocity, $\omega$ is the robots angular velocity, $\dot{\phi}_{L}$ and $\dot{\phi}_{R}$ are the robot's left and right wheel speeds, $r$ is the radius of each of the robot's wheels, and $l$ is half of the robot's wheelbase. The kinematics are then defined as


Figure A.1: Closed loop control diagram

$$
\left[\begin{array}{c}
\dot{x}  \tag{A.3}\\
\dot{y} \\
\dot{\theta}
\end{array}\right]=\left[\begin{array}{cc}
\cos (\theta) & 0 \\
\sin (\theta) & 0 \\
0 & 1
\end{array}\right]\left[\begin{array}{l}
v \\
\omega
\end{array}\right],
$$

where $x$ and $y$ denote the robot's global position, and $\theta$ denotes the robot's orientation. We want to control the wheel speeds such that the robot follows a smooth curve. We define a linear control law that considers the distance of the goal location as well as the current and desired orientations of the robot. Let

$$
\begin{gather*}
v=k_{\rho} \rho,  \tag{A.4}\\
\omega=k_{\alpha} \alpha+k_{\beta} \beta \tag{A.5}
\end{gather*}
$$

where $\rho, \alpha$ and $\beta$ are found in Figure A. 1 and $k_{\rho}, k_{\alpha}$ and $k_{\beta}$ are our controller gains. By adjusting these $k$ gains, we can tune the robot to behave desirably.

## Appendix B

## The A* Algorithm

The A* algorithm is used to compute a path from one position to another. We assume that a robot is trying to get to a goal. To compute a path using the $A^{*}$ algorithm, the robot creates a binary map $P$ which contains every cell $p$ in $\mathcal{W}$. Each $p_{i}$ which lies in $\mathcal{S}$ is given a value of 1 , and the rest are given values of 0 . This matrix is known as an occupancy grid. We define $\mathbf{p}_{e}$ as the list of every $p_{i}$ with a value of 0 . The robot next assigns a heuristic value $h_{i}$ to every $p_{i}$ in $\mathbf{p}_{e}$. We define this value simply as the euclidean distance from that $p_{i}$ to the desired goal position $p_{g}$.

Beginning with the starting position, $p_{s}$ the robot checks all adjacent cells to see if they are contained in $\mathbf{p}_{e}$. If they are, then they are added to a list of cells that potentially lie on the desired path. This list may simply be called $O$, the open list. For each position added to $O$, the current cell being evaluated is denoted as the parent cell to each of these adjacent cells. We call the list of parent cells $D$. This is the most important step, since $D$ will eventually consist of the path from $p_{g}$ to $p_{s}$.

Once these steps have been completed, the current $p_{i}$ is added to another list $K$, called the closed list. $K$ is simply a list of cells that have already been visited. When a cell is added to $K$, it is removed from $O$. To pick the next cell to evaluate, we want to choose the cell from $O$ that has the lowest $f_{i}$ value, where

$$
\begin{equation*}
f_{i}=h_{i}+g_{i} . \tag{B.1}
\end{equation*}
$$

Here, $g_{i}$ is the distance to $p_{i}$ from the $p_{s}$, and $h_{i}$ is the distance from $p_{i}$ to $p_{g}$. Since we don't know the actual distance from $p_{i}$ to $p_{g}, h_{i}$ is just an estimate. The new $p_{i}$ is picked and the process continues until the $p_{g}$ is found. If an adjacent cell is found that is already in $O$, then it must be determined which parent cell is better for that particular cell. Once the $p_{g}$ is added to $K$, the search has been completed. The path from the $p_{g}$ to $p_{s}$ is $D$ going backwards from $p_{g}$ to $p_{s}$. Figure B. 1 shows an example of the $A^{*}$ algorithm finding a path to the goal. For this example, $D$ would be $N-J-I-E-A$.


$$
\begin{aligned}
& \text { Heuristics } \\
& A=14, B=10, C=8, D=6, E=8, F=7, G=6 \\
& H=8, \quad I=5, J=2, K=2, L=6, M=2, N=0
\end{aligned}
$$

Figure B.1: An example solution to $A^{*}$. The heuristics are the $h_{i}$ scores for each spot. [3]

## Appendix C

## Voronoi Decompositions

According to [20], a Voronoi diagram is "the partitioning of a plane with $n$ points into convex polygons such that each polygon contains exactly one generating point and every point in a given polygon is closer to its generating point than any other." The diagram $G_{V}$ itself consists of Voronoi edges, $E_{V}$, which denote the boundaries of each cell, and Voronoi vertices $V_{V}$, which denote every intersection of $E_{V}$. Therefore, a Voronoi cell $C_{V}^{i}$ is represented by a set of vertices $\left\{v_{V}^{1}, \ldots, v_{V}^{m}\right\}$ in $V_{V}$ and the set of edges $\left\{e_{V}^{1}, \ldots, e_{V}^{m}\right\}$ in $E_{V}$ that connect them. The size and shape of each $C_{V_{i}}$ is determined by its generating Voronoi point $\mathbf{p}_{i}$ according to

$$
\begin{equation*}
C_{V}^{i}=\left\{\mathbf{q} \in \mathcal{W}\| \| \mathbf{q}-\mathbf{p}_{i}\|\leq\| \mathbf{q}-\mathbf{p}_{j} \|, \forall j \neq i\right\} \tag{C.1}
\end{equation*}
$$

where $\mathcal{W}$ represents the workspace, $\mathbf{q}$ represents a point in $\mathcal{W}$, and $\left\{\mathbf{p}_{1}, \ldots, \mathbf{p}_{n}\right\}$ represents the set of all Voronoi points in $\mathcal{W}$. The target density function $\phi_{t}$ is the density of the building material, which is static during construction. These diagrams provide a simple way of dividing the workspace. Figure C. 1 shows a Voronoi diagram.

There are several ways to generate Voronoi diagrams. One easy way is to use a computer program that has a built-in Voronoi diagram generator. To gain a better


Figure C.1: An example of a Voronoi diagram
understanding of how Voronoi diagrams are actually decomposed, a simulation of Fortune's sweepline algorithm was created. Figure C. 2 shows a step by step breakdown of how the algorithm works.

In Figure C.2(a), we define the Voronoi points in the workspace $\mathcal{W}$. We then begin to sweep a horizontal line $H$ down the graph starting from the top. In Figure C.2(b), $H$ crosses the first $\mathbf{p}_{i}$. Upon encountering a point, a vertical line emanating from the point is formed. As the sweepline continues downward, the vertical line becomes a parabola $P$ with the $\mathbf{p}_{i}$ as the vertex and $H$ as the directrix, or $P_{i}=\left\{\rho_{1}, \ldots, \rho_{n}\right\}$ where

$$
\begin{equation*}
\rho_{i}=\left\{\mathbf{q} \in \mathcal{W}\| \| \mathbf{q}-\mathbf{p}_{i}\|=\| \mathbf{q}-H_{x} \|\right\} \tag{C.2}
\end{equation*}
$$

where $H_{i}$ point on $H$ with the same $x$ coordinate as $\mathbf{q}$. A parabola $P_{i}$ is formed every
time the sweepline encounters a $\mathbf{p}_{i}$.
Once more than one parabola exists, the two parabolas will intersect with each other twice (unless the two points lie at the same $y$ coordinate, in which case they will only intersect once, but this is a special case). We let $V_{P}$ denote the set of these intersection points, where $V_{P}=\left\{v_{P i j}^{1}, v_{P i j}^{2}, \ldots, v_{P m n}^{1}, v_{P m n}^{2}\right\}$. We see this occur in Figure C.2(c). Each $E_{V}$ is formed by drawing a line from $v_{P i j}^{1}$ to $v_{P i j}^{2}$. We can only see one of the intersection points in this figure. The beachline $B$ is the part of the system of parabolas that is closest to the sweepline, as seen here:

$$
\begin{equation*}
B=\left\{\rho_{i} \mid\left\|\rho_{i}-H_{x}\right\|<\left\|\rho_{j}-H_{x}\right\|\right\} \tag{C.3}
\end{equation*}
$$

Based on Equation C.3, there will only be one $\rho$ for every $x$ coordinate. The beachline in Figure C. 2 is clearly represented by bold pink dots. The remaining sections of the parabolas that do not lie on the beachline are shown as regular pink dots.

Figure C.2(d) depicts the third $\mathbf{p}_{i}$ crossing. As $H$ continues down, we can see the formation of the first $v_{V}$ by the two rightmost $\rho$ points as they converge. It is at this point where one of the parabolas that was once a part of the beachline is no longer a part of the beachline. Figure C.2(e) depicts a few moments after this has happened. The black dot represents $v_{V}^{1}$ created by the sweepline. A new edge is now formed between this $v_{V}^{1}$ and the new intersection of the two parabolas $v_{P}$. The moment before the last Voronoi vertex is formed can be found in Figure C.2(f).

In theory, $H$ could continue down the graph forever, since some of the edges will extend to infinity. This is obviously not necessary, though, and $H$ is stopped once every $v_{V}$ has been found and the edges extend a reasonable length such that the shape of the decomposition can be easily seen [1].


Figure C.2: Fortune's Sweepline Algorithm

## Appendix D

## The Brushfire Algorithm

The brushfire algorithm provides a clearance map of the area, where the clearance from the edge of the map and the border of the structure are considered. To understand how the brushfire algorithm works, we refer to Figure D.1. In this figure, we see numbers growing as they get farther from the structure and the edge of the workspace.


Figure D.1: Brushfire algorithm applied to the outer border of a structure and the edges of the map.

## Bibliography

[1] D. Ausitin. Voronoi diagrams and a day at the beach. American Mathematical Society Feature Column: Monthly Essays on Mathematical Topics. url: http://www.ams.org/samplings/feature-column/fcarc-voronoi.
[2] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. Principles of Robot Motion: Theory, Algorithms and Implementation. MIT Press, Cambridge, MA USA, 2004.
[3] H. Choset, A. Mills-Tettey, V. L.-S. Jr., P. N. Atkar, and K. Tantisevi. Lecture notes from robotic motion planning: A* and d* search. Robotics Institute at Carnegie Mellon University 16-735.
[4] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. Transactions on Robotics and Automation, 20(2):243-255, April 2004.
[5] H. Flanders. Differentiation under the integral sign. The American Mathematical Monthly, 80(6), June - July 1973.
[6] M. A. Hsieh. Motion Control Strategies for Networked Robot Teams in Environments with Obstacles. PhD thesis, University of Pennsylvania, 2007.
[7] M. A. Hsieh, V. Kumar, and C. J. Taylor. Constructing radio signal strength maps with multiple robots. In Proc. of the Int. Conf. on Robotics \& Automation (ICRA'04), 2004.
[8] M. A. Hsieh and J. Rogoff. Complexity measures for distributed assembly tasks. In Proc. of Performance Metrics for Intelligent Systems Workshop (PerMIS '10), Baltimore, MD, USA, September 2010.
[9] C. Hu, W. Chen, Y. Chen, and D. Liu. Adaptive kalman filtering for vehicle navigation. Journal of Global Positioning Systems (2003), 2(1), November 2003.
[10] E. Klavins. Programmable self-assembly. Control Systems Magazine, 28(4):43 56, August 2007.
[11] K. Lee, M. Moses, and G. S. Chirikjian. Robotic self-replication in structured environments: Physical demonstrations and complexity measures. The International Journal of Robotics Research (IJRR '08), 27(3-4), March - April 2008.
[12] L. Pimenta, M. Schwager, Q. Lindsey, V. Kumar, and D. Rus. Simultaneous coverage and tracking (scat) of moving targets with robot networks. In Proc. of the 8th Int. Workshop on the Algorithmic Foundation of Robotics, Guanajuato, Mexico, December 2008.
[13] A. C. Sanderson. Parts entropy methods for robotic assembly system design. In Proc. of the Int. Conf. on Robotics $\mathcal{E}^{\mathcal{G}}$ Automation (ICRA'84), March 1984.
[14] A. C. Sanderson, L. S. H. de Mello, and H. Zhang. Assembly sequence planning. AI Magazine, 11(1):62-81, Spring 1990.
[15] C. Schuil, M. Valente, J. Werfel, and R. Nagpal. Collective construction using lego robots. In Proc. of National Conference on Artificial Intelligence (AAAI '06), Boston, MA., USA, July 2006.
[16] M. Schwager, D. Rus, and J.-J. Slotine. Decentralized, adaptive control for coverage with networked robots. International Journal of Robotics Research, 28(3):357-375, March 2009.
[17] Y. Terada and S. Murata. Automatic modular assembly system and its distributed control. International Journal of Robotics Research (IJRR '08), 27(3-4):445-462, March - April 2008.
[18] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. International Journal of Robotics Research (IJRR '01), 20(5), 2001.
[19] S. Thrun, W. Burgard, and D. Fox. Probabilistic Robotics. MIT Press, Cambridge, MA USA, 2005.
[20] E. W. Weisstein. Voronoi diagram. url: http://mathworld.wolfram.com/VoronoiDiagram.html.
[21] J. Werfel, Y. Bar-Yam, and R. Nagpal. Building patterned structures with robot swarms. In Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05), pages 1495-1502, Pasadena, CA USA, July 2005.
[22] J. Werfel, Y. Bar-Yam, D. Rus, and R. Nagpal. Distributed construction by mobile robots with enhanced building blocks. In Proc. of the Int. Conf. on Robotics $\xi^{G}$ Automation (ICRA'06), pages 2787-2794, Orlando, FL USA, May 2006.
[23] J. Werfel, D. Ingber, and R. Nagpal. Collective construction of environmentallyadaptive structures. In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'07), pages 2345-2352, San Diego, CA USA, October 2007.
[24] J. Werfel and R. Nagpal. Three-dimensional construction with mobile robots and modular blocks. The International Journal of Robotics Research (IJRR '08), 27(3-4):463-479, March/April 2008.
[25] J. Worcester, J. Rogoff, and M. A. Hsieh. Constrained task partitioning for distributed assembly. In Submitted to the Conference on Intelligent Robots and Systems (IROS '11).
[26] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems - challenges and opportunities for the future. IEEE Robotics and Autonomation Magazine, March 2007.
[27] S.-K. Yun and D. Rus. Adaptation to robot failures and shape change in decentralized construction. In Proc. of the Int. Conf. on Robotics 83 Automation (ICRA'10), pages 2451-2458, Anchorage, AK, USA, May 2010.
[28] S.-K. 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, Lucerne, Switzerland, Aug-Sept 2009.

