Optimal Self Assembly of Modular Manipulators with Active and Passive Modules

by

Seung-kook Yun

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

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2009

Author .....
Department of Electrical Engineering and Computer Science
Jan 06, 2009

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

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

Optimal Self Assembly of Modular Manipulators with Active and Passive Modules

by

Seung-kook Yun

Submitted to the Department of Electrical Engineering and Computer Science
on Jan 06, 2009, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In this thesis, we describe algorithms to build self-assembling robot systems composed of active modular robots and passive bars. The robotic module is the Shady3D robot and the passive component is a rigid bar with embedded IR LEDs. We propose algorithms that demonstrate the cooperative aggregation of modular robotic manipulators with greater capability and workspace out of these two types of elements. The distributed algorithms are based on locally optimal matching. We demonstrate how to build an active structure by the cooperative aggregation and disassembly of modular robotic manipulators. A target structure is modeled as a dynamic graph. We prove that the same optimality - quadratic competitive ratio - as for the static graph can be achieved for the algorithms. We demonstrate how this algorithm can be used to build truss-like structures. We present results from physical experiments in which two 3DOF Shady3D robots and one rigid bar coordinate to self-assemble into a 6DOF manipulator. We then demonstrate cooperative algorithms for forward and inverse kinematics, grasping, and mobility with this arm.

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

Acknowledgments

I am deeply grateful to my advisor, Daniela Rus. She has been an excellent mentor and a great guider for my research. I would like to thank to Marsette Vona, Mac Schwarger, Carrick Detweiler and Iuliu Vaslescu for their help in every stage of this work. I also appreciate the members of Distributed Robotics Lab.

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of the thesis is to develop algorithms for self-assembly of truss climbing robot Shady3D and passive elements, and to execute manipulation tasks by the assembled structure. We introduce a system where two robots can be assembled by using a passive strut between them. We wish that the algorithms are distributed and as optimal as possible. The thesis approaches the problem by three steps:

- Truss navigation of multi modular robots by locally optimal matching
- Self-assembly by extending the truss navigation algorithm
- Reconfiguration of the self-assembled structure

We consider robot motions for self-assembly as navigation on robot elements as well as on truss. First we propose distributed algorithms for locally optimal truss navigation of multi truss climbing robots without self-assembly. Then we extend the algorithms to self-assembly by considering movement of self-assembly as navigation on the robots. The proposed algorithms are implemented in simulations and experiments. After self-assembly, manipulation tasks are executed by reconfiguring the self-assembled structure.

1.1 Truss Navigation of Multi Truss Climbing Robots

We continue the study of truss climbing robots began in [46] and consider coordination problems when multiple robots are tasked to do work on the truss. Each robot is allocated

a different location on the truss. We wish to develop a distributed algorithm that uses local information only (e.g. sensing and communicating locally) to plan paths for each robot from their initial locations to the target locations. We consider a set of identical robots that are capable to navigate a 3D truss-like structure such as the Shady3d robots in [46]. The key technical challenge is to plan an optimal set of collision-free paths that minimize the number of steps (and therefore energy consumption) for each robot. Collisions occur when one robot unit blocks the way, as these robots can only travel on free truss segments. Since the robots have information about their own mission only, it is very likely that the robots may encounter other robots along the way and need to look for alternative paths.

This problem arises within construction and inspection applications. Trusses are encountered as part of bridges, scaffoldings, space structures, and underwater platforms such as oil rigs. Tasks related to trusses are often dangerous and difficult for human workers, as the bars are narrow. Space construction and maintenance outside a spacecraft require dangerous extravehicular activity (EVA) missions by astronauts. We wish to create truss-climbing robots can do significant work to inspect, augment, or repair engineered truss structures. In the more distant future, these robots might become capable of climbing natural structures, such as trees, to assist with agricultural applications.

Coordinating a group of robots moving on a truss is easy when all the information about the environment, the robots, and their goals is available centrally. We can represent the truss as a graph, whose vertices are attachment places for the robots on the truss and edges connect adjacent links. Then the problem of moving k robots to their goal location along optimal collision-free paths reduces to a min-cost disjoint path problem with vertex capacities (since at one time each vertex can be occupied by only one robot.) This becomes an evacuation or assignment problem and has been studied extensively, for example optimal time and cost solutions are presented in [21]. The solution intuition is as follows: the truss graph can be transformed into a directed graph by connecting a super source to every starting node, a super sink to every target, and a vertex for each intersection of two paths from starting to target. Careful expansion of a min-cost max flow algorithm with unit capacity such as [15] will produce optimal vertex-disjoint paths with the cost of a min-cost bipartite matching. The running time of one of the best algorithms is $O(k(k^2+n+$

$m)log(n+k^2))$, where m is number of edges in a graph and n is number of vertices.

While simple, the centralized solution to this problem does not capture the reality of k robots moving autonomously and independently on a 3D truss to perform individual work at different locations. We wish to develop a distributed algorithm that relies on local information only, that can be realistically sensed and communicated by the robots. In the thesis, we describe a distributed planning algorithm for placing k identical robots on a 3D truss. We assume that the truss geometry is known to each robot and that the robot can detect and communicate with other robots located at neighboring nodes (e.g. one edge away). We describe how sensing and communication can be used to guarantee that robots travel to their goal locations in an optimal number of steps. We analyze the running time of this algorithm and the competitive ratio. We show that our algorithm has quadratic competitive ratio and compare the result to a greedy algorithm whose competitive ratio is exponential. Finally, we present data from extensive simulations and from several physical experiments with Shady3D Robots.

1.2 Self-assembly by Locally Optimal Matching

Given a framework for truss navigation and truss climbing robots, we wish to provide self assembling capabilities to such a system. In other word, we would like to have the robot elements grasp materials such as bars from the world and self-assemble as a truss objects with desired geometry. We consider this problem when the robot elements are the same robots we developed for climbing. They work with rigid passive bars that are augmented with communication capabilities to aid the robots with locating and grasping them.

More specifically, we wish to develop modular robots capable of construction tasks that integrate robotic elements and raw materials from the environment to create dynamic and controllable complex objects. In our previous work [7, 42] we describe a mechanism and supporting algorithm for the self-assembly of linkages that alternate 3 DOF robot modules called Shady3D with rigid bars. The resulting assemblies are controllable using distributed inverse kinematics protocols to achieve pick and place tasks. We extend the algorithm for self-assembling linkages in [42] to the self-assembly of arbitrary truss structures consisting

of rigid bars and Shady3D-like robots with 3 rotational DOFs that are capable of grasping the bars on both ends. We assume a cache of robots and a cache of rigid bars. The robots know the goal shape but they do not know about each other. They are only capable of detecting each other and communicating locally, when they are in close proximity of each other. We show that this problem can be reduced to a distributed matching problem and analyze how sensing and communication can be used to guarantee that the robots construct the goal structure in an optimal number of steps. We then describe an implementation of this algorithm in simulation. Discussing its performance with a physical system will be considered in future.

The robot abstraction used by this algorithm is modeled on the Shady3D robot [7]. We assume that the robot looks like a rigid bar. The robot has grippers at both ends and is capable of grasping both rigid bars and robot units. The grippers can rotate. An additional rotational degree of freedom in the middle of the robot allows it to twist. This type of robot uses its grippers as feet to move in a truss-like environment. The truss provides the *grounding support* for each robot and for the truss-like assemblies the robots can create by grasping rigid bars. Thus, the scope of this work is restricted to truss construction in truss-like environments. Applications range from self-assembling scaffolds for construction to underwater bridges and space structures.

1.3 Manipulation Tasks with Self-assembled Arm

Next, we build on the truss self-assembly results to explore the development of low-cost modular manipulators. Drawing from the theoretical, practical, and existing experience in manipulation and modular robotics, we propose an approach to synthesize modular manipulators that match a desired workspace by self-assembly. We envision robot systems capable of scavenging raw materials from the environment to adaptively create dynamic programmable structures that integrate robotic elements with passive components. We describe how a collection of simple robotic modules can grasp rigid bars and coordinate to self-assembled robotic manipulators with a higher number of degrees of freedom and a larger workspace than the components. The resulting robot arms are distributed mobile

manipulation systems that can be controlled to accomplish the basic functionality of a robot arm: inverse kinematics, forward kinematics, grasping, and pick and place. These arms can move autonomously to different places in the workspace. The specific type of arm we study alternates robotic elements with rigid bars. The presence of the rigid bars enhances the structural rigidity of the system and also contributes to the total number of degrees of freedom of the system. The total number of elements is determined by the required workspace size. We aim to synthesize the smallest robot structure that meets the workspace requirements.

The robot arms in this work belong to a class of robots called active linkages, that were introduced in our previous work [7]. Active linkage robots look like trusses and are comprised of two types of modules: passive structural modules which may either be fixed in the environment or free to move individually, and mobile active modules which may pick up or climb on the passive modules, organize and hold them in a desired shape, and actively move them for self-assembly, self-reconfiguration, or self-repair purposes. The passive modules can be passed around by the active modules and coordinated to form the skeleton of a large class of truss geometries. The active modules can also be thought of as smart joints in the linkage.

The challenge in building self-assembled modular arms ranges from issues related to designing simple and robust active modules capable of interacting with other passive and active modules, to problems of control and planning. Control is challenging because each active link is a separate robot. The many degrees of freedom of these systems have to be coordinated using distributed and efficient controllers.

More specifically, we present algorithms for the self-assembly of multi-link robot arms out of 3DOF robot modules with the structure and capabilities of our robot Shady3d [47] and rigid bars with embedded LEDs for guiding grasping. We assume to know the location of the robot modules and of a cache of smart passive bars. Given a desired workspace, we determine the number of needed links. A distributed self-assembly algorithm constructs the robot arm as an alternation of robot elements and passive bars. We demonstrate this algorithm in the context of creating a 6DOF manipulator out of two Shady3D elements and one passive bar. We also present cooperative algorithms for forward and inverse kine-

matics, grasping, and pick and place and give data from physical experiments. Finally, we demonstrate that this type of modular arm is mobile and can move autonomously to a different location in the workspace.

1.4 Summary of Contribution

Contributions of the thesis are summarized. A new framework for self-assembly of truss climbing robots is introduced. Active and passive modules for assembly are designed and implemented. Next, fully distributed control laws are developed so that the robot elements navigate on truss as well as assemble themselves into a given structure. The control algorithms are proven to have the local optimum that has quadratic competitive ratio to the global optimum. Finally, The algorithms are implemented on the proposed system with various tasks.

1.5 Organization

This thesis is organized as follows. Chapter 2 shows previous work of truss climbing robots, self-assembly of modular robots and matching algorithms in a graph. Our system with active robot elements and passive bars with embedded IR LEDs are introduced in Chapter 3. Theories and implementation of the truss navigation is described in Chapter 4. Self-assembly based on the extended locally optimal matching is explained and implemented in Chapter 5. Chapter 6 shows experiments of self-assembly and manipulation tasks. We conclude the thesis in Chapter 7

Chapter 2

Related Work

The idea of robots which self-assemble (and/or self-replicate) using elements from the environment is not new, for example see Chirikjian et al's paper on lunar-surface self-assembly [20] and references therein. In this paper we explore the particular idea of separating the system into active modules and passive bars, with an emphasis on the possibility only producing the latter—much simpler—units from the environment.

Our proposed systems and algorithms are further related to prior work in the fields of self-reconfiguring robots, hyper-redundant robots, variable-geometry truss robots, and truss climbing robots.

2.1 Self-Reconfiguring Robots

Of all the self-reconfiguring modular robots which have been previously reported, our current work seems most closely allied with systems based on rotary DOF and mechanical connection mechanisms.

Normally, the robots have a form of lattice or chain. Also, they can be classified by a number of unit types in a system. A homogeneous system has a single unit, and a heterogeneous system has two types of units. A homogeneous unit always comes with an actuator, whereas a heterogeneous system may include a non-movable unit for a battery or any bulky material.

Because of nature of modular robots, mostly each system requires a unique controller

for reconfiguration, according to its structure. Distributed controllers are prefered in a sense that a large number of the modules may need to work together.

Murata, et al's built "3D Fracta" [40] which works like a reconfigurable lattice. The robot unit has rotatable connectors on each side of a cube so that it can move another unit. A stochastic algorithm is used to control the units in a distributed way.

Kotay and Rus developed "Molecule"[31, 30, 29] which has male and female connectors to assemble it to another molecule and can lift up the connected molecule in 3D. The proposed controllers move a group of the molecules in a distributed fashion.

Rus and Vona built "Crystal"[10, 11, 12, 9] which expands and shrinks its body for 2D reconfiguration. They introduced an algorithm to move a cube from one location to another in a distributed way.

Unsal, Kiliccote, and Khosla made bi-partite "I-Cubes" [8] system which is heterogeneous with a cubic module and a link module. Centralized locomotion algorithms were used with given combinations of the modules.

Lund, Beck, Dalgaard, Støy et al developed ATRON [22, 27] which is a sphere rather than a lattice. Each unit has an upper and lower hemisphere and the structure lead to a complicated controller for 3D reconfigration.

Duff, Yim, et al's PolyBot [13] is a chain-type module, and linked modules can reconfigure themselves to an arbitrary 3D chain. They showed how tens of the modules are cordinated to change a global structure such as from a four-legged robot to a snake or a fully connected chain.

A major difference in our present work is that we are proposing modular systems with only some modules containing active DOF—the rest serve as passive structural elements. In contrast, all of the above referenced systems are either homogeneous (all modules identical and actuated) or are heterogeneous but still require actuation in all modules.

2.2 Hyper-Redundant Robots

Research in the field of hyper-redundant robots has mainly explored non-reconfiguring systems with high DOF and fixed kinematic topology, typically open chains. Both planar

systems and full spatial mechanisms have been explored. The planar systems typically have one (effective) kinematic DOF per link, and the spatial systems may have two or more. Sometimes the links are internally parallel mechanisms, an arrangement which has been called "hybrid serial-parallel" [16, 43, 24].

Burdick and Chirikjian built "snakey" (which is also a variable geometry truss, see below) [18, 19] that has 10 modules each of which has 3 prismatic joints. Total 30 degree-of-freedom enables the manipulator to avoid obstacles in their experiments as if it is flexible continuum.

Greenfield, Rizzi, Choset et al designed a modular snake[3], which can have many links and climb through a pipe by bracing its body. A controller of the robot works with ambiguous fiction and dynamics.

Suthakorn and Chirikjian built a binary-actuation manipulator. The manipulator has four of 3-bit planar VGT modules each of which has 8 possible reachable points. The module has discrete rotary joints with three binary actuators. They also proposed a numerical and analytical ways of inverse kinematics.

Wolf, Choset, et al's "Schmoopie" [2] was built for search and rescue missions. The robot arm has 14 actuated universal joints in a chain.

Our proposed two-leg tower construction is a hybrid serial-parallel mechanism; and our single-chain tower is kinematically equivalent to typical hyper-redundant snakes. Thus far we have applied classical pseudoinverse-derived inverse kinematics methods for these structures but we are also considering adaptation of methods developed specifically for hyper-redundant robots, for example Chirikjian's "backbone curve" method [17].

2.3 Variable Geometry Truss and Truss Climbing Robots

Variable geometry trusses (VGTs), can be viewed as a generalization of the serial-chain hyper-redundant systems to more general kinematic topologies. Both fixed-topology systems like the NASA/DOE "SERS DM" [39] and manually-reconfigurable systems—notably Hamlin, Sanderson, et al's TETROBOT [16]—have been considered. Also related are robotic systems which assemble static trusses, for example, Everest, Shen, et al's SO-

LAR [25], and Howe and Gibson's "Trigon" system [1]. Such self-assembling and self-reconfiguring truss systems are a promising direction for robotic assembly of large structures in space—for example, see Doggett's overview of automatic structural assembly for NASA [44].

Truss *climbing* robots are also under active investigation, e.g. Amano et al's handrail-gripping robot for firefighting [23], Ripin et al's pole climbing robot [48], Nechba, Xu, Brown et al's "mobile space manipulator SM2" [36, 37], and Almonacid et al's parallel mechanism for climbing on pipe-like structures [34]. Truss climbing also has been acknowledged to have clear applications in inspection and construction of in-space structures [4].

Several truss climbing robots have been explored by other groups, e.g. Staritz et al's "Skyworker" [38], Amano et al's handrail-gripping robot for firefighting [23], Ripin et al's pole climbing robot [48], Nechba, Xu, Brown et al's "mobile space manipulator SM2" [37], Kotay and Rus' "Inchworm" [28], and Almonacid et al's parallel mechanism for climbing on pipe-like structures [34].

This paper presents a new mechanical design and novel control using intentional mechanical compliances and proprioception, with experimentally confirmed robustness.

Our proposed systems can act as self-reconfiguring/self-assembling modular VGTs, and our Shady3D robot shows how the same module designs can also be applied to truss-climbing.

2.4 Matching Algorithm

In truss navigation, we deploy robots to target points on truss by minimum number of total moves to minimize the energy usage. Collision-free min-cost path planning algorithms[15] for multi-robots are required since a robot can not move through the other robot. The problem can be solved by a perfect matching between initial nodes and target nodes, with selection criteria for distributed controllers. The matching uses the fact that the robots are identical. This is not an offline matching problem, but closer to the online matching. However, this problem is different from the online matching in that a robot does not know how

other robots are matched. In the online matching, it is proven that the lower bound of the competitive ratio of any deterministic algorithm is $(2k-1)$. The permutation algorithm[26] achieves this bound.

Chapter 3

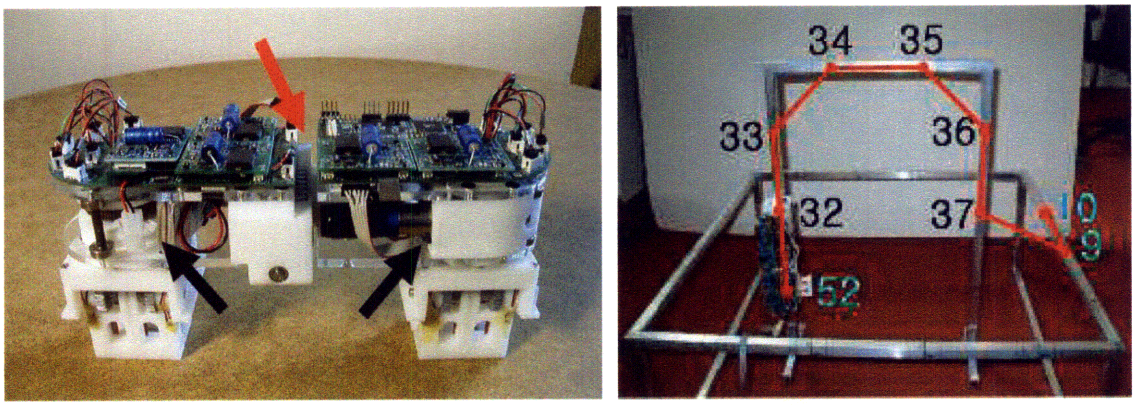
Experimental infrastructure: Shady3D and rigid bars with LEDs

In this work, we use Shady3D [47] as an active module and a bar with embedded IR LEDs as a passive one. The resulting arm can be anchored anywhere on truss. The algorithms presented here depend on the abstract capabilities of Shady3D and can be instantiated on any other robot module with similar capabilities. We introduce the hardware and how they build a self-assembled tower.

3.1 Shady3D

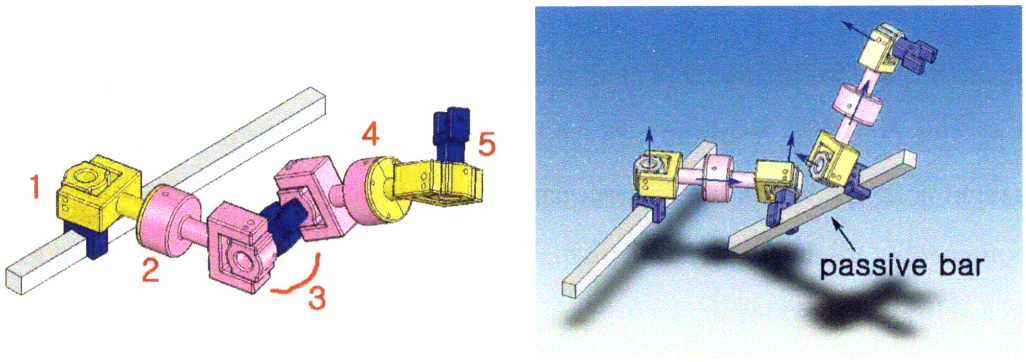
Shady3D was originally designed with the goal of climbing 3-dimensional trusses as a first step toward tree-climbing robots. It has three joints for 3-D motion and two grippers on each side as shown in Figure 3-1. The number of joints is chosen to be minimal for moving on the 3-D trusses. Unlike Shady [14] which was designed to climb planar trusses, the middle joint enables Shady3D to switch from one plane to another. The robot can only reach specific points, where the trusses are modeled by nodes and edges [47], and every robot has the identical structure and functions.

The three joints of Shady3D enable a robot to traverse 3D trusses. By connecting two Shady3Ds (See Figure 3-2(a)) directly we can generate a 5DOF linkage. The DOF is not six due to the fact that the axes of two gripper joints lie on the same line. A 6DOF linkage



(a) (b)

Figure 3-1: (a) Shady3D robot and its structure: 3-joints and 2 grippers (b) truss structure and an example of deployment of a single robot: numbers denote nodes on the trusses[45]

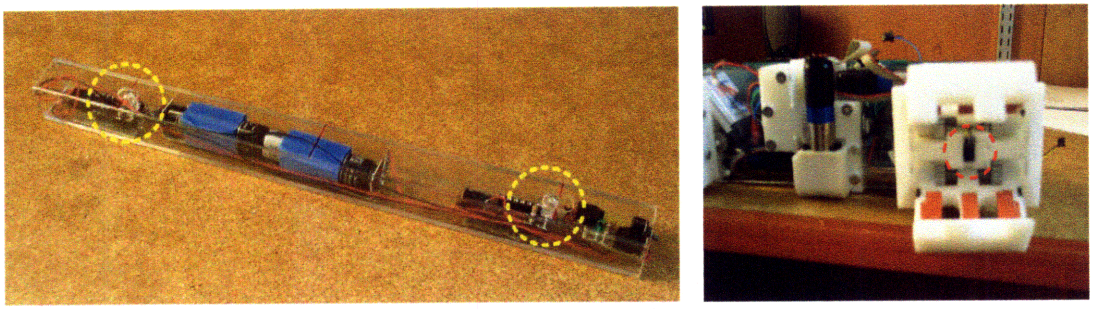


(a) (b)

Figure 3-2: (a) A 5DOF manipulator with directly connected two Shady3Ds (b) A 6DOF one with inserting a passive bar between two robots[45]

is obtained by using a truss element as a medium of connection as in Figure 3-2(b). Since the two robots are grasping different points of the passive truss, the reduction of DOFs does not happen.

We have built two fully working Shady3D robots and 5 Shady3d bodies that do not include any electronics, but can be used as obstacles during our experiments, to simulate the presence of up to 7 robots working together on the truss. The placement algorithms are implemented and tested using this environment.



(a) (b)

Figure 3-3: (a) A passive bar with embedded IR LEDs (b) an IR sensor attached beneath the gripper

3.2 Passive bar

The self-assembly operation requires many grasping steps that need to be robust. We choose an approach that embeds beacons in the passive object. Solutions that rely on other sensors such as vision are possible but require more computation. A passive bar emits IR signals via the IR LEDs embedded in the bar is shown in Figure 3-3(a). Two LEDs are located at each side of the bar (indicated by yellow dotted circles) and inform a robot about existence of the bar. Each bar includes two AAA batteries as a power source. Shady3D has two IR sensors on each gripper bed, as shown in Figure 3-3(b), so that it can check whether a bar is present just below its gripper or not. In our experiment, it can sense the bar located about 50(cm) below the gripper. Note that this sensing range matters when a combined 6DOF manipulator also tries to find the bar, because it is capable of moving any direction while a single module can not.

Witha a map, the robot is able to predict where to find a bar and it can confirm by sensing the IR signal. Note that it cannot be sure there is a bar even if it knows its location, because the robots work in a distributed fashion and another robot might have grasped and moved the bar.

Chapter 4

Truss Navigation

4.1 Problem Formulation

Consider a truss with k robots on it. Each robot receives a set of goal locations. In this section we formulate the assumptions in the distributed placement problem we wish to solve. Local information only will be used to direct each robot to a goal location so that all targets are guaranteed to be occupied. We make the following assumption and notations:

- We are given a 3D truss with known geometry. The robots can grasp the truss at a discrete set of points. The truss is modeled as an undirected graph G, whose vertices are points where a robot can grasp (for example, the joint of the truss is not a vertex) and whose edges connect adjacent vertices. There is a positive cost on each edge.
- Each robot is modeled as one point that corresponds to an anchor gripper on the graph G.
- There are k identical robots on the truss.
- The robots can sense if an adjacent vertex is free or occupied by another robot.
- Each robot can communicate with the robots that occupy adjacent nodes.
- When two robots communicate, they can share all information (e.g. state, target location, etc.)

- The set of initial nodes in the graph is R; robot i is initially located at r_i. These locations are not known to the robots.
- The set of target nodes is T; $T = \{t_1, t_2, ..., t_k\}$
- The cost of a set of paths is the sum-total of the edge weights of the paths.
- The goal is for all target nodes to be occupied by the robots and for the overall path cost to be minimal.

4.2 Distributed Algorithm by Locally Optimal Matching

In this section we describe a solution to the distributed placement of k robots on a truss using distributed locally optimal matching. The intuition behind this solution is that robots compute the location of the nearest target using as input the truss geometry and the list of targets. Then they start traveling toward their target in parallel. If the path of a robot is blocked by a different unit, or the robot finds that its target is already occupied, the robot is reconfigured by an operation of swapping state. This solution can be described as finding a matching between R and T.

Each robot runs a local algorithm for planning a path and moving along the path. The robot algorithm consists of several phases: initialization, path computation, path execution, and path reconfiguration in case of deadlock. Each robot's state includes the following data:

- *ID*: identification number
- *Status*: what it is doing now.
- *Settle Down*: true if it has settled down at the target
- *Pushing List*: a list of the robots pushing it now
- *Location*: currently occupying nodes
- *Initial and Target node*

- *matching list*: a list of the initial and target nodes a robot has learned by the collisions only with *settled-down* robots.
- *Path to the target*: a list of the nodes to the target

The following sections detail the phases of the algorithm.

4.2.1 Initialization

Using the initial state, the truss geometry, and the given set of targets, the robot computes the nearest target node.

4.2.2 Deployment

Algorithm 1 shows the main path execution loop. Since robots do not know where the other robots are, they need to be able to detect collisions with other robots.

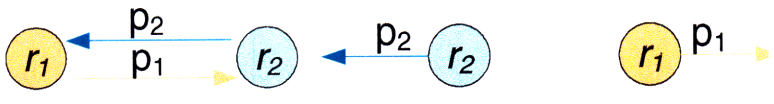
Algorithm 1 Main Control

```
1: while true do
2:    Status=Planning
3:    if not Communicating then
4:       Distributed Deployment
5:    end if
6:    Status = Idle
7:    Wait for Communication in a fixed time
8: end while
```

Algorithm 2 shows the *Distributed Deployment* procedure. This enables a robot to advance, to detect collisions, and to handle collisions. First, the algorithm checks if this robot has arrived at its desired target. If so, it sets the resource *Settle Down* as *true*, and stops. Otherwise, the robot checks for collisions by send a message (note that the system is set up so that a robot can only communicate with adjacent robots and only when the status of the robots is *Idle*.) If the next node is empty, it takes a step and updates the resources. : it changes the location and makes the *Pushing List* null. Flushing the pushing list indicates that the robot is not within a any cycle. In case of a collision the robot determines the collision type (one of five cases) and takes corresponding action as shown in Section4.2.3.

Algorithm 2 Distributed Deployment

```
1: if has reached my target then
2:    Settle Down = true
3: else
4:    Communicate with adjacent robots
5:    if Next node empty then
6:       Move to the next node
7:    else
8:       switch Cases of the Collision
9:          case The paths are crossing
10:            Exchange the robots
11:         case The blocking robot has not settled down
12:            Add Pushlist(my PushList + my ID)
13:            if ID∈PushList & ID=min(PushList) then
14:               Exchange the robots
15:            end if
16:            Wait until it moves while pushing it
17:         case settled down and the distinct target
18:            Exchange the robots
19:         case settled down and non-anchor is occupying
20:            request Step Aside
21:         case settled down and the same target
22:            Merge matching information of the robots
23:            Find a new locally optimal matching
24:      end switch
25:   end if
26: end if
```



(a) (b)

Figure 4-1: Exchanging robots; r is a robot and p is a path (a) two crossing paths and (b) the system state after the exchange

4.2.3 Handling Collisions

Crossing path

When the path of a robot and the blocking robot cross each other, the two robots exchange their destinations by exchanging all their state (see Figure 4-1.) Upon completing state exchange by communication, the robots return to their *Distributed Deployment* algorithm and eventually diverge as shown in Figure 4-1(b).

Breaking a Deadlock

A deadlock is a status in which some robots can not move even though their paths do not cross as shown in Figure 4-2. There are four robots $\{r_1, r_2, r_3, r_4\}$. Their paths $\{p_1, p_2, p_3, p_4\}$, form a rectangular cycle (see Figure 4-2(a)). The *Pushing List* (which consists of robots waiting to advance) is used to eliminate deadlock. Each blocked robot sends its list to the blocking robots. The blocker merges the list onto its own. When a robot finds itself on its Pushing List (as shown in Figure 4-2(b)), it is in a cycle. In this case, the robot with lowest ID forces the blocking robot to execute *Exchanging the robots* until a robot in the list does not block it anymore. After this forced exchange, the cycle will be broken as shown in Figure 4-2(c). The robot with the lowest ID can proceed.

Stepping aside

Stepping Aside is necessary because Shady3D has two grippers and occupies two nodes. Consider the scenario in Figure 4-3(a). The green robot has settled down at the node occupied by the right gripper (the anchor), and the blue robot is trying to go through the node occupied by the non-anchor gripper of the green robot. Even though the paths do not cross,

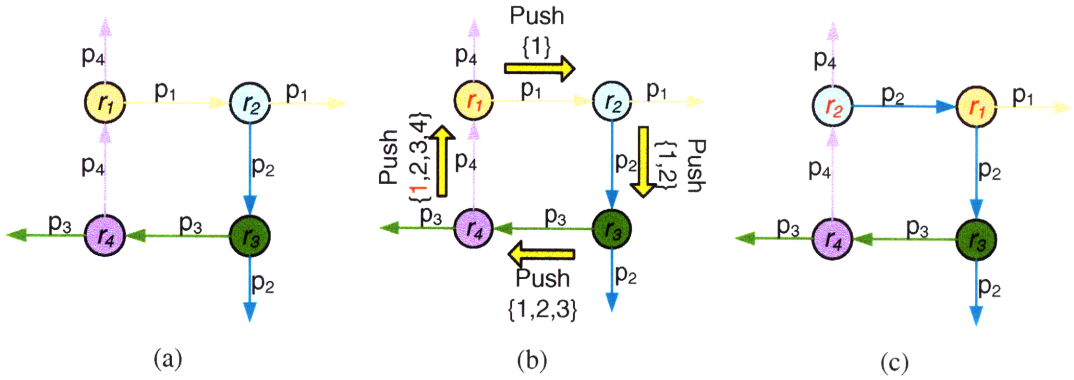


Figure 4-2: Breaking a deadlock: (a) a deadlock (a cycle) (b) communication protocol for preventing the deadlock: *push* (c) the cycle is broken by exchanging identities.

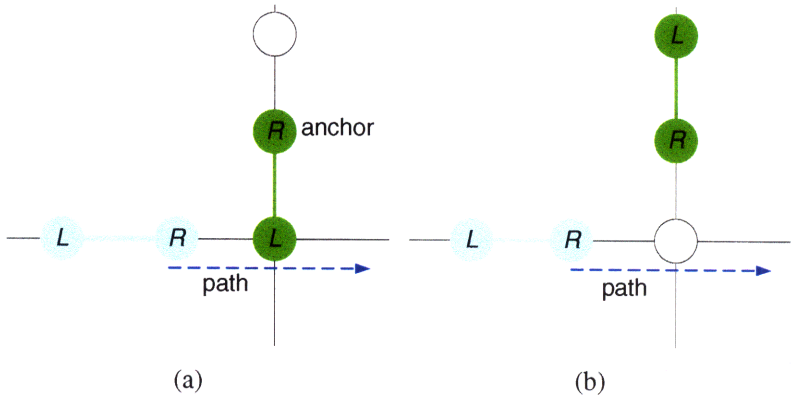


Figure 4-3: Stepping Aside: (a) the blue robot requests stepping aside of the green robot (b) after the stepping aside of the green robot

there is no cycle, and the two robots have the distinct targets, the blue robot is stuck. In this case the blocked robot requests the blocking robot to step aside so it can move as shown in Figure 4-3(b). If there is no room for the non-anchor gripper to move to, the blocking robot communicates this failure and *Exchanging the robots* eliminates the deadlock.

Finding a new target

The last selection criterion explains how to select a new target when two robots that block each other have the same target node. To solve this conflict, the blocked robot finds another target node, computes a new path to it, and follows the path as it has done. The new target is a node in the optimal set of the target nodes matched with the initial nodes of the blocked

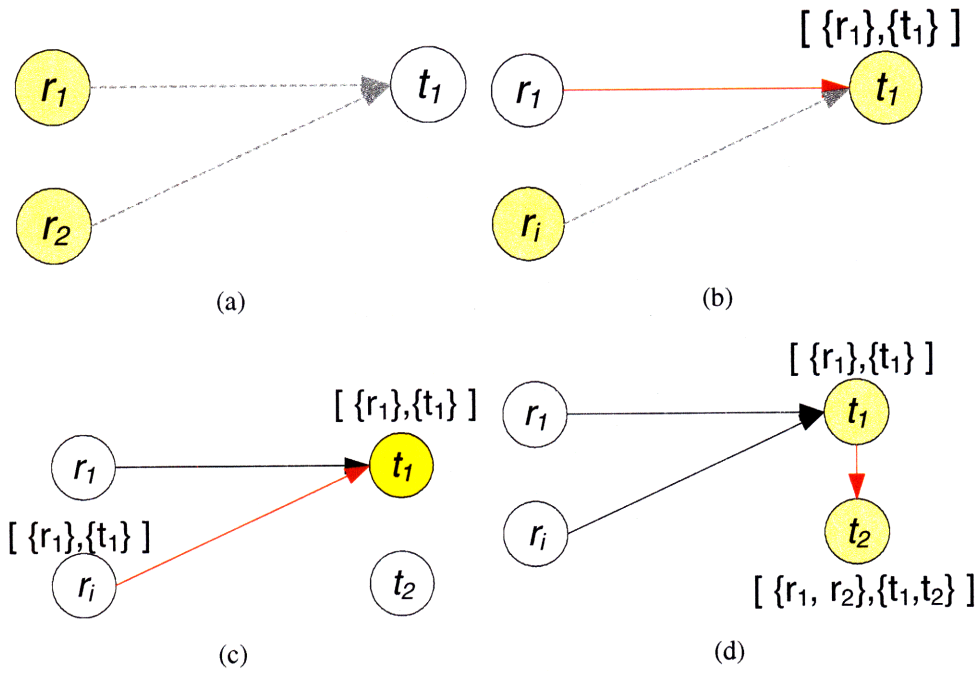


Figure 4-4: Two robots are deploying according to the proposed matching algorithm: (a) initially, robot #1 and #2 have the same target node by the local optimal matching (b) Robot #1 arrives earlier than #2, and now it has its matching list $[\{r_1\},\{t_1\}]$. (c) Robot #2 finds out t_1 has already been occupied upon its arriving at t_1. (d) Robot #2 gets a new target t_2 by calculating the locally optimal matching with $[\{r_1,r_2\},\{t_1\}]$. After arriving t_2, it updates the matching list by $[\{r_1,r_2\},\{t_1,t_2\}]$

robot and the Matching lists of both robots.

Algorithm 3 describes the locally optimal matching. When a robot p has found that the other q occupied its target, they merge their Matching list $\{R_q,T_q\}$ and $\{R_p,T_p\}$. Using the merged Matching list and its starting node, it calculates the next target which is incident on the locally optimal matching M_{p+q}. This is computed efficiently using the *Hungarian algorithm*[32] which has $O(k^3)$ runtime and can be used with any size matrix.

Algorithm 3 Getting a Locally Optimal Matching

1: Merge the *matching lists* $\{R_q,T_q\}$ and $\{R_p,T_p\}$
2: Find the next target which belongs to M_{p+q} by *Hungarian algorithm*
3: Change *matching lists* of both robots to the merged one

4.2.4 Termination

If at least one robot is moving, the total distance between the robots and the target nodes is strictly decreasing. If each robot has a distinct target node the perfect matching has been reached. Since there is no deadlock in the system the actions of the robots will converge and terminate.

4.3 Analysis

In this section, we analyze the optimality and the computational runtime of the distributed matching algorithm for placing robots of a truss. We show that the distributed algorithm has $O(k^2)$ asymptotic competitive ratio to the global optimum and the total runtime of all robots is bounded by $O(k^5 + k^2(n+m)logn)$.

The distributed localized robot placement problem is different from the online matching problem in that robots do not know how other robots have already been matched to requests. The lower bound of the competitive ratio of any deterministic on-line matching algorithm is $(2k-1)$. The permutation algorithm[26] achieves this bound. Furthermore, our solution is more efficient than a greedy one.

4.3.1 Online matching: previous work

Our analysis uses several results on minimal weight partial matching from [26]. The minimal weight partial matching M_i which is the set of edges that form the minimal weight partial perfect matching between the subset $\{r_1, r_2, ..., r_i\}$ and subset of T with a minimal number of edges in $M_i - M_{i-1}$. Let T_i be the subset of T consisting of vertices of T which are incident on M.

Lemma 1 *The cost of the M_is form a monotonically non-decreasing sequence.*

Lemma 2 *For each i, the set difference $T_i - T_{i-1}$ contains exactly one vertex.*

Lemma 3 *For a union set composed of vertices of M_{i-1} and $\{r_i, t_i\}$ where r_i is chosen to be a incident vertex on M_i, the cost from r_i to t_i is bounded by the cost of $M_{i-1} + M_i$*

Proofs of Lemma 1- 3 are given in [26]. From now, We also use M as the cost of the set M.

In this section we extend the partial optimal matching results to our algorithm. Let $R_p \in R$ contain the robots that settled down on their target nodes according to robot p. Let M_{R_p} be the min-cost matching of the sub-group R_p. Note that M_{R_p} is sequentially constructed as a new robot collides with it. Therefore, Lemma 1- 3 hold for R_p and M_{R_p}.

4.3.2 Running Time

In this section we examine the running time of our algorithm by first showing that the a match between an initial robot and a target location does not change once a robot settles on the target, and then examining the size of the merged matching lists.

Suppose each robot has the cost matrix C where C_{ij} is the cost of the shortest path from r_i to t_j. The running time required to compute this matrix is $O(k(n+m)logn)$ (k times the running time of a shortest-path algorithm runtime. We use the partial matrix of $C(R_{p+q}, T)$ corresponding to a set of $\{R_p \cup R_q \cup initialnode\}$ and entire T. After getting the new target, the matching lists are exchanged with the merged list, and the robot follows a new path to the target. It is important that the new pair of the initial and the target node not be included in the merged list, because other collisions may cause a change in the target node.

Lemma 4 *The initial node and target node matched by a settled down robot stays fixed.*

Proof: *Suppose a robot starting at r_a has arrived at its target t_a and settled down. This robot will leave the target node, if and only if another robot exchanges the resources with it, by Algorithm 2. This can only happen for type 3 collisions. In this case, the exchange of states between the two robots causes the matched pair (r_a, t_a) to be maintained.*

Lemma 5 *A merged matching list from the lists of two robots has the exactly same number of initial nodes and target nodes.*

Proof: *Suppose that robots A and B have their matching lists $\{R_a, T_a\}$ and $\{R_b, T_b\}$ which have the same number of nodes in both R and T. If A has the same target as B, they collide. Suppose B is settled. The merged list is obtained as $\{R_a \cup R_b, T_a \cup T_b\}$. Therefore, the*

number of the initial nodes and the target nodes is:

$$N_R = n(R_a) + n(R_b) - n(R_a \cap R_b) \tag{4.1}$$

$$N_T = n(T_a) + n(T_b) - n(T_a \cap T_b) \tag{4.2}$$

,where a funcion $n()$ notifies the number of a set. Since we know that each matching list has the same number of R and T, the result is true if and only if:

$$n(R_a \cap R_b) = n(T_a \cap T_b), \tag{4.3}$$

where a funcion $n()$ notifies the number of a set. By Lemma 4, Equation 4.3 holds, because otherwise at least one of the initial nodes in $\{R_a \cap R_b\}$ must match more than one target nodes.

Lemmas4 and 5 imply that it is necessary to add only one target node to $\{T_a \cap T_b\}$ in order to match $\{R_a \cap R_b, r_a\}$. Figure 4-5 shows how a robot behaves using the algorithm. In Figure 4-5(a), robot p with matching list $\{R_p, T_p\}$ collides with robot q with $\{R_q, T_q\}$ at the node t_q. Then p gets a new target and the merged list $\{R_p \cup R_q, T_p \cup T_q\}$. Then p follows a new path from t_q to t_{p+q}. Note that the path to t_q from somewhere among T_p belongs to the previous step. We have $O((p+q)^3)$ running time by Hungarian algorithm[32].

4.3.3 Optimality by Competitive Ratio

In this section we investigate optimality by finding the competitive ratio, which is defined as the cost of the worst case to the globally optimal one.

Lemma 6 *When robot p with matching list $\{R_p, T_p\}$ collides with robot q with matching list $\{R_q, T_q\}$ at node t_q(See Figure 4-5), the path to the new target t_{p+q} costs up to $4M_{p+q}$, where M_{p+q} is the optimal matching between the set $\{R_p \cup R_q, r_p\}$ and $\{T_p \cup T_q, t_{p+q}\}$.*

Proof: *The path is the red line in Figure 4-5(a), from t_q to t_{p+q}. By triangular inequality (Figure 4-5(b)), the cost is bounded by the edges, (t_q, r_p) and (r_p, t_{p+q}).*

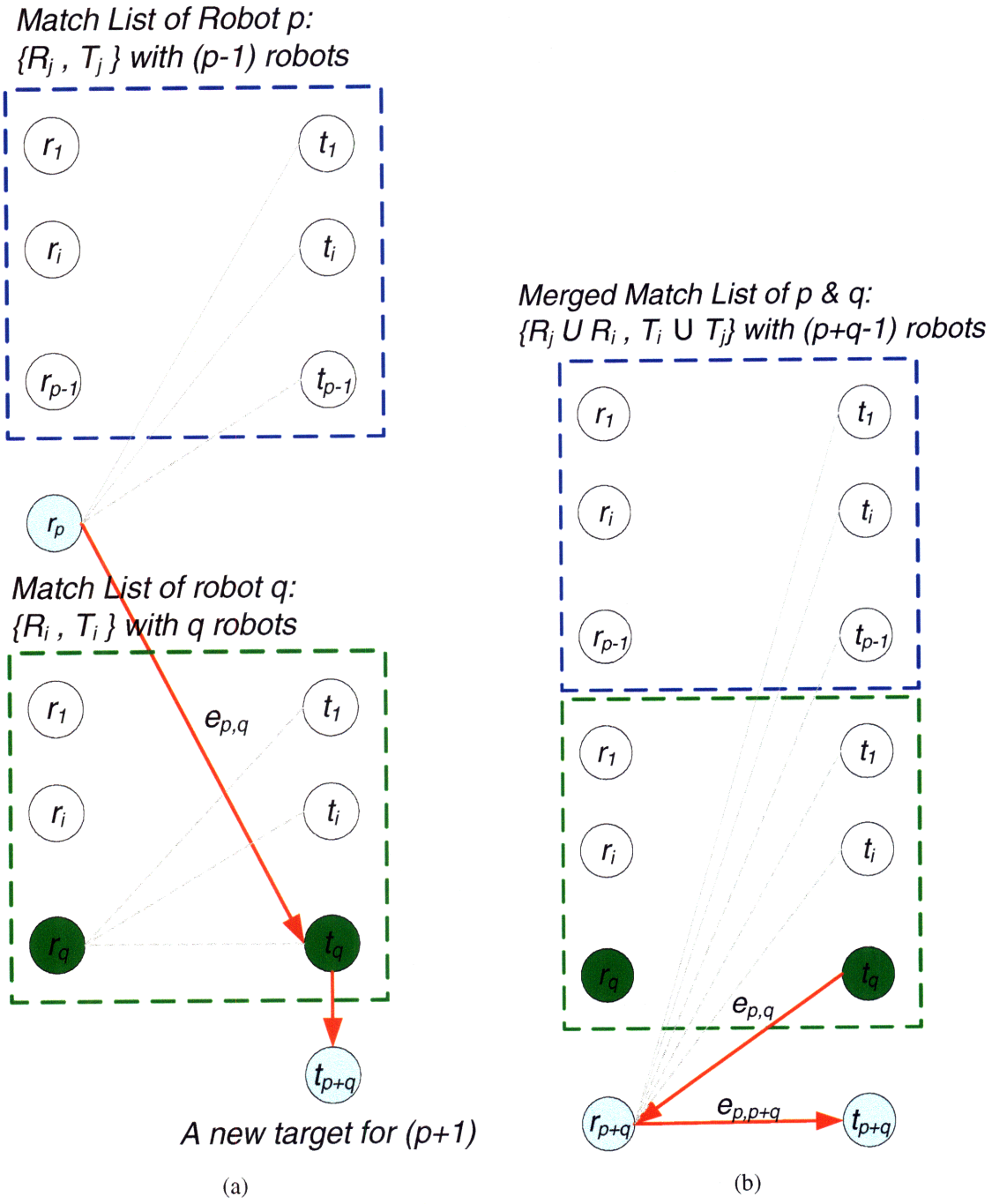


Figure 4-5: The procedure of the deployment

$$cost(t_q \rightarrow t_{p+q}) \leq e_{p,q} + e_{p,p+q} \tag{4.4}$$

The first edge is due to the previous collision and bounded by $M_{p-1} + M_p$, which is the locally optimal matching between $\{R_p, T_p\}$ and $\{\{R_p, r_p\}, \{T_p, t_q\}\}$ respectively, by Lemma 3. The second is also bounded by $M_{p-1+q} + M_{p+q}$, the locally optimal matching of $\{R_p \cup R_q, T_p \cup T_q\}$ and $\{\{R_p \cup R_q, r_p\}, \{T_p \cup T_q, t_{p+q}\}\}$. Therefore, Equation 4.4 becomes:

$$\begin{aligned} cost(t_q \rightarrow t_{p+q}) &\leq M_{p-1} + M_p + M_{p-1+q} + Mp + q \\ &\leq 4M_{p+q} \end{aligned} \tag{4.5}$$

because the cost of the local optimum never decreases by Lemma 1.

Lemma 7 *The total cost of the path for the i^{th} robot to the settled down position is bounded by $(4i-3)M_{R_i}$.*

Proof: *We use induction. For convenience, assume that the robots deploy sequentially, one by one after the previous one's settlement. When the first robot settles down, the lemma holds. Suppose the bound is true for $(i-1)$, and then the cost P_{i-1} is:*

$$P_{i-1} \leq (4(i-1)-3)M_{R_{i-1}} \tag{4.6}$$

By lemma 6, P_i is bounded by following:

$$\begin{aligned} P_i &\leq P_{i-1} + 4M_{R_i} \\ &\leq (4(i-1)-3)M_{R_{i-1}} + 4M_{R_i} \\ &\leq (4i-3)M_{R_i} \end{aligned} \tag{4.7}$$

Intuitively, we can consider the result as a function of the number of collisions. The maximum of i-th robot is $(i-1)$, since there are $(i-1)$ robots before it. Every time it collides, the added cost is bounded by Lemma 6, and it leads to the above inequality.

Lemma 8 *The distributed deployment algorithm has $O(k^2)$ asymptotical competitive ratio.*

Proof: *The total cost of deployment is the sum of all k robots' cost, and it can be written as follows:*

$$\begin{aligned} \sum_{i=1}^{k}(4i-3)M_{R_i} &\leq \sum_{i=1}^{k}(4i-3)M_R \\ &= (2k^2-k)M_R \end{aligned} \tag{4.8}$$

where M_R is the global optimum.

One added complication is the cost of stepping aside. This cost does not relate with any optimal cost, it remains as a constant. As R gets bigger, we can Therefore, competitive ratio is $(2k^2-k)$, asymptotically.

Lemma 9 *The total running time of the distributed placement algorithm is $O(k^5+k^2(n+m)logn)$.*

Proof: *Whenever a robot is getting a new target by the proposed algorithm, $O(k^3)$ runtime is required. Therefore, total runtime for Hungarian algorithm is $O(k^4)$. $O(k(n+m)logn)$ is to calculate the cost matrix. Total runtime of all robots is obtained to multiply k to the runtime of each robot $O(k^4+k(n+m)logn)$.*

Thus, our proposed distributed matching algorithm has $(2k^2-k)$competitive ratio, and $O(k^2/logk)$ times the running time of the central controller.

The quadratic competitive ratio is due to the fact that the robots do not share their matching lists. If a robot can share the information with the collided robots by any method, we achieve $O(k)$ competitive ratio, which is the competitive ratio of the online matching algorithm.

4.3.4 Comparison to The Greedy algorithm

In this section we show that the performance of a distributed greedy algorithm has an exponential competitive ratio and thus much worse than our algorithm.

Consider an intuitive greedy algorithm where each robot successively finds the nearest target node. Consider Figure 4-6 where $(k-1)$ robots have occupied $(k-1)$ target nodes

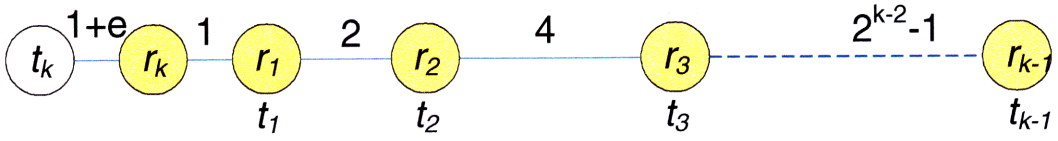


Figure 4-6: A bad situation for the greedy algorithm.

and one robot must find its target by the greedy algorithm. If the left edge of the robot has $(1+\varepsilon)$ cost, where ε is a small positive number, the robot will continuously go to the right node and finally return to t_k after visiting all the right nodes. The cost is (2^k-1), while the optimal cost is only $(1+\varepsilon)$. Therefore, in this case, the competitive ratio is (2^k-1) when ε is small.

4.3.5 Discussion

By this reason, we are quite sure that the ratio should be worse than that of the online matching. However, it seems hard to prove the exact bound while the online matching is easily proven to have $(2k-1)$ for any deterministic algorithm[26].

Could randomized algorithm help?

An alternative to improve performance is to consider a randomized version of the algorithm. The best competitive ratio of the randomized algorithm[35] is $O(log_3 k)$ while a deterministic is $O(k)$. They use a special graph, HST tree, rather than a generic graph. In addition, they have shown that a generic graph can be modified into a HST tree. We have not tried this approach to our problem. However, we conjecture that a randomized algorithm would not work well in our case, because it is much harder to narrow the probability - the choices. For instance, for a uniform metric graph where nodes are fully connected and every edge cost is 1, the proposed randomized online matching algorithm - which finds the nearest one and if there is tie (multi nodes with the same cost), it randomly select one - gives $O(logk)$ expected competitive ratio while a greedy one does $O(k)$. However, in our case, the same algorithm also yields $O(k)$, which is the same order as that of a greedy one. Although one example cannot tell everything, we guess this is a hint that a randomized

algorithm might not work better mainly because there is no central brain, which know the current state of all robots, and it makes it impossible to avoid the collision.

4.4 Implementation

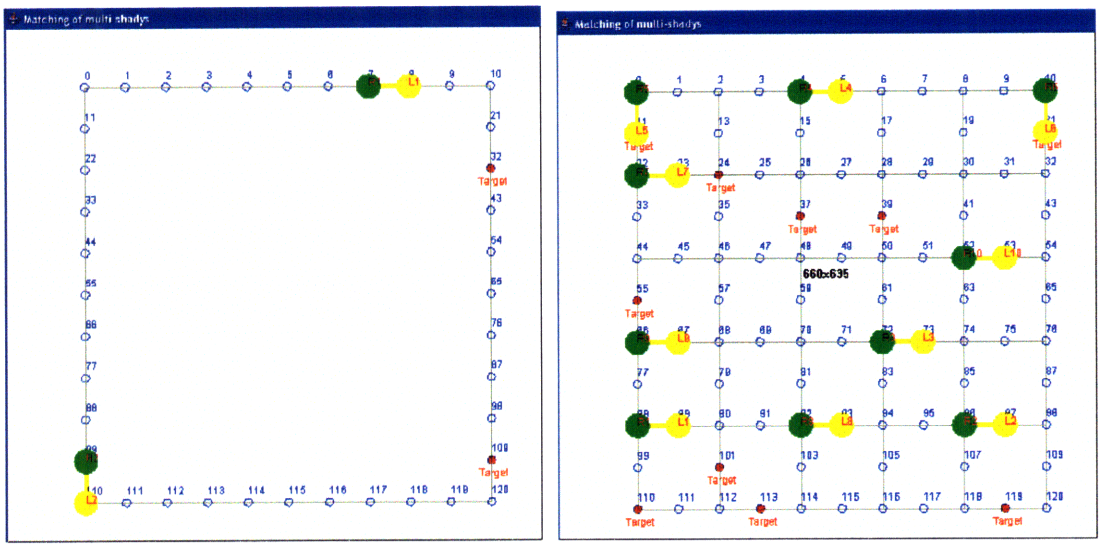
We implemented the distributed deployment algorithm in simulation and on the physical platforms described in Section 3.

4.4.1 Simulation

We have implemented the distributed placement algorithm in Java. We simulated each robot as an independent process (thread) to ensure parallelism. The target nodes and the initial placement of the left and right grippers of each robot were randomly selected. The left gripper was initially used as the robot's anchor.

We have tested two kinds of geometries and generated sparse and dense graphs, as shown in Figure 4-7. For each graph, 2$\sim$10 robots are simulated 100 times, respectively. The parameter of the fixed communication time was set 0.1 second. The number of the communications and the faster total execution time are inversely proportional to this time. The total time is defined as the duration from the start of the simulation till the termination of each robot process.

The statistical results collected from these simulations are shown in Table 4.1. For both graphs, the average ratio of the cost by the distributed algorithm to the cost of the centralized globally optimum algorithm are very slowly proportional to the number of the robots, whereas the analytical worst bound increases in a quadratic fashion. Even with 10 robots, the ratio is only around two. It appears that the graph type does not affect it. The average number of communication per robot is larger in the sparse graph than in the dense graph. This makes sense because a robot tends to have more chances to collide on a sparse graph.



(a) (b)

Figure 4-7: (a) Sparse graph with two robots (b) Dense graph with ten robots

Table 4.1: Result of the simulations

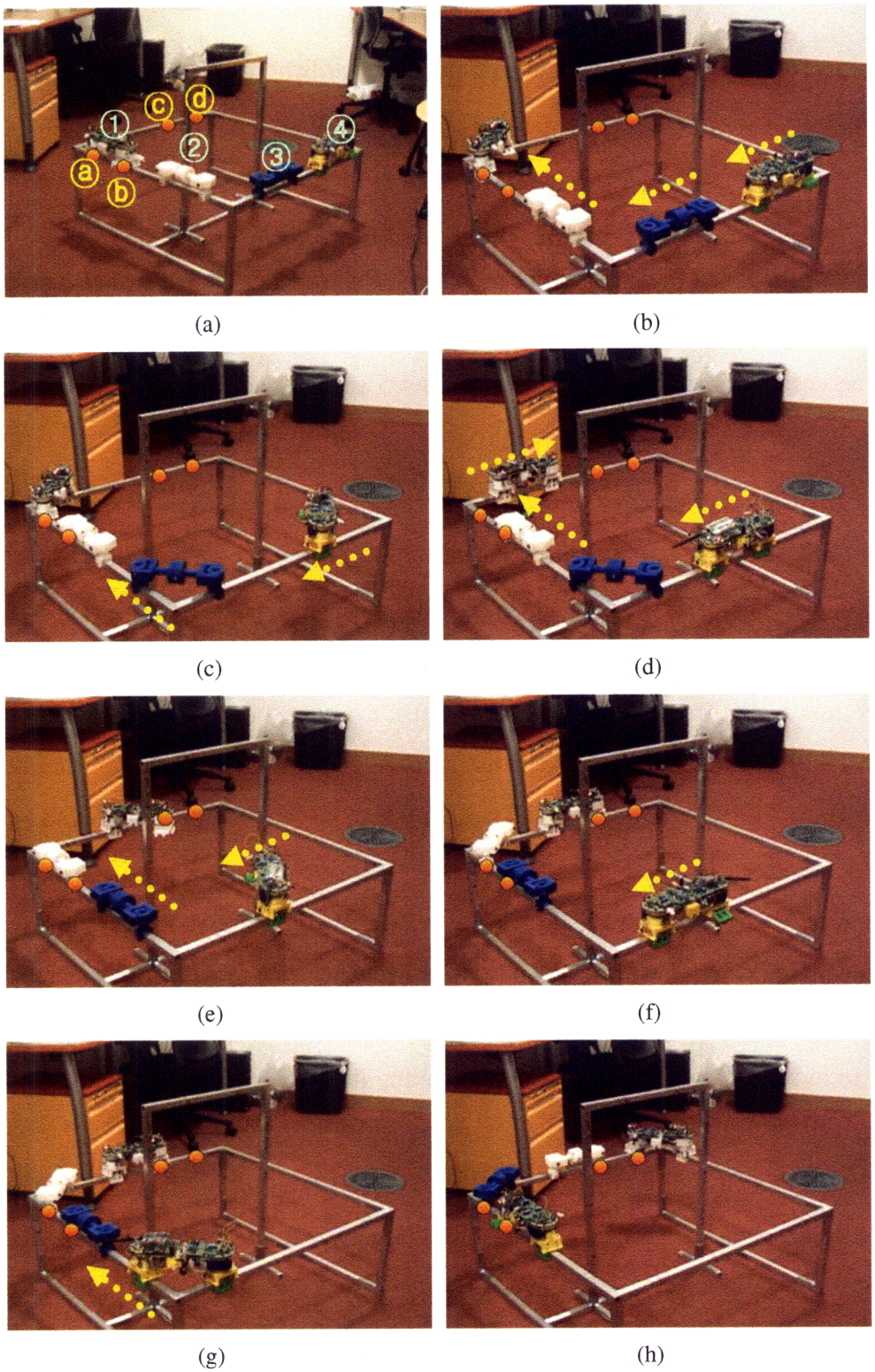
# of robots	Avg. Ratio Real/Opt		Avg. Comm. per robot		Worst Cast Real/Opt	
	Sparse	Dense	Sparse	Dense	Sparse	Dense
2	1.1	1.1	0.7	0.8	1.8	1.7
3	1.2	1.3	1.3	1.8	2.1	2.8
4	1.3	1.4	1.5	2.7	2.4	2.6
5	1.5	1.6	3.0	3.9	2.5	3.3
6	1.6	1.8	4.4	5.1	2.6	3.4
7	1.8	1.7	4.2	3.0	2.9	3.2
8	1.9	1.8	4.9	3.9	3.5	3.1
9	1.9	1.9	5.8	3.9	4.4	4.1
10	2.0	2.1	6.2	5.0	3.3	3.4

4.4.2 Physical Experiment

The simulation algorithm has been transfered to the hardware system in Section 3. For these experiments we use two Shady3D robots, two inert Shady3D robot bodies, and a simple truss structure as shown in Figure 4-8. Shady3D communicates via Bluetooth receivers. Its range is adjusted to the one-edge distance. The inert robots are introduced to increase the number of robots and collisions in this setup (at the moment we have only two Shady3D robots.) They are manually controlled and simulated by the main computer as if they move and even communicate by themselves. We set 5 seconds as communication rate and 10 seconds as the moving time of the inert robots (the human operator moves them during this time).

We performed six experiments using different robot placements and target locations. Snapshots of the experiment are shown in Figure 4-8, where four robots are moving and orange circles represent the target nodes. Four target nodes $\{a,b,c,d\}$ are displayed as orange circles. In the beginning (Figure 4-8(a)), every robot finds its nearest target node. We see robot #1 is already located at its target a, and it does not move. The other three robots plan to move to the same target node b which has already been occupied by #1. By the proposed algorithm, #2 communicates with #1 and requests step aside to free the target. Consequently, #1 steps away (Figure 4-8(b)). After #2 settles on b, #3 follows its way and collides with #2 (Figure 4-8(c)). Because they have the same target, #3 calculates the local optimal matching based on the merged known list, and it heads to the result, target c. Then it tries to move to c, and finds that #2 is blocking its way. They exchanges the resources. Successively, #2 also exchanges the resources with #1. In the end, #1-3 robots move to c, a, and b respectively, and they all settle down (Figure 4-8(f)). Next #4 is moves to b (Figure 4-8(g)). #4 gets a new target d using locally optimal matching with the merged known list of #3 and #4. In the same way, each robot moves, shifting to the next target, which is caused by successive exchanges. Finally each robot reaches at one of the target nodes (Figure 4-8(h))

The performance summary for the six physical experiments is given in Table 4.2. While the competitive ratio is almost the same as for the simulation case, the average communi-



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 4-8: snapshots of reconfiguration of four robots

Table 4.2: Result of the Experiments

Exp #	Optimal Cost	Traveled Cost	Exchange Count	Avg. Comm.	Finding new Opts
1	15	15	9	22	3
2	14	25	4	5	5
3	19	20	3	3	3
4	4	4	0	0	0
5	17	17	2	13	3
6	28	28	14	14	3

cation is much higher. Note that the number of communication is highly dependent on the communication rate.

We have encountered errors in the experiments. The errors are caused by over current of the motors in case of misalignment of the robot gripper and the truss- while moving. We are working on improving the robustness of the hardware.

Chapter 5

Self-assembly by Locally Optimal Matching

5.1 Self-assembled linkage: walking tower

Multiple Shady robots can connect to one another using passive bars to form a larger active structure. The robots become smart joints in the self-assembled structure: they can actuate the structure to travel, bend, twist, and self-reconfigure. Figure 5-1 shows snapshots of the self-assembly of a truss tower. Twelve active modules and eight passive bars are employed to build a three-dimensional tower that can reconfigure itself by controlling active parts. Note that the robots are controlled by just a given sequence of motions - designed by hand. We will implement the same structure in Section 4.4 by the proposed algorithm in a distributed way.

5.2 Problem Formulation

Our goal is to build an active structure composed of Shady3D-like robots and passive bars. We extend the algorithms we proposed in [41], where we consider how to optimally place a group of robots on a truss, to how the robots can create truss-like structures by self-assembly. Local information only will be used to coordinate robots to reach their designated good locations.

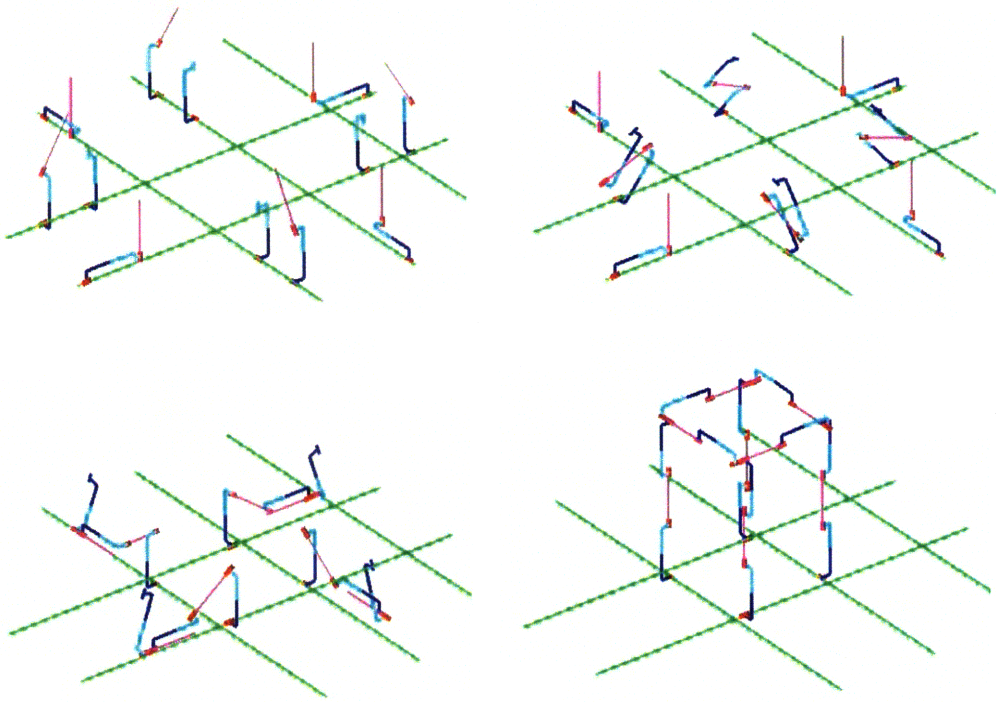


Figure 5-1: Four snapshots of the tower building simulation. The Shady3D robot modules are drawn as an elongated U-shapes with light and dark halves; the free bars and the grid are drawn as straight segments [45].

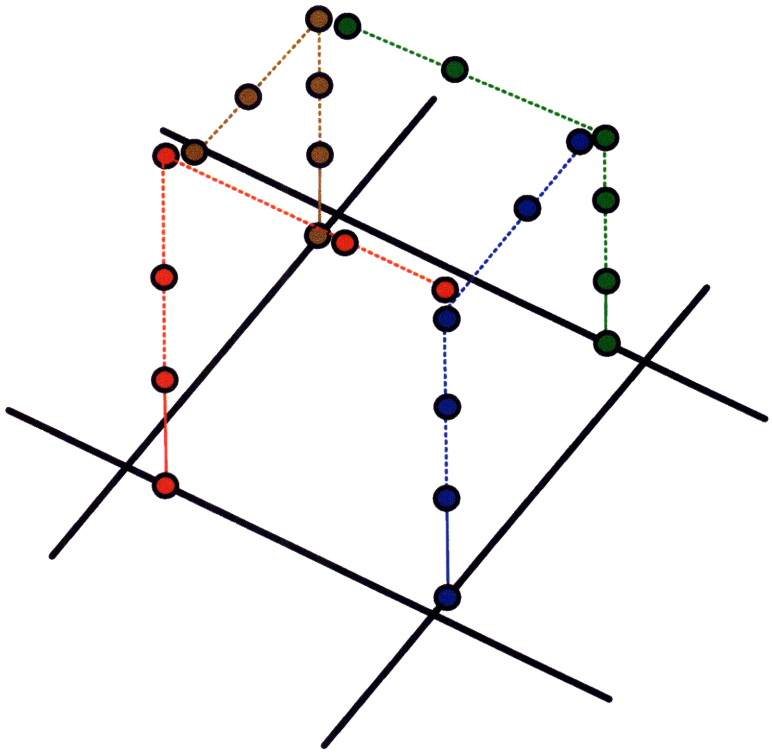


Figure 5-2: A graph representation of the tower. Only the target nodes are shown. Circles are nodes, and non-activated and activated edges connect them. The graph has 4 on-truss and 16 non-truss target nodes. Each color denotes one of four trees.

Let A be a target to be assembled, A can be represented as a graph as shown in Figure 5-2. The graph has active (solid lines in Figure 5-2) and non-active (dotted lines in Figure 5-2) edges. The nodes that are reachable to the robots are connected by active edges. The nodes have two types: a truss node is a part of the existing scaffold while a non-truss node is at the growing robotic truss. The graph begins as a connected system that marks the initial truss/scaffold for the assembly. As a robot learns that some of the non-truss nodes are occupied, its graph is updated by adding edges connected to the nodes. The more it learns, the more nodes become reachable. Every non-truss node should be linked from a on-truss node by non-activated edges. We call such a node (on a truss) a *root*, and the nodes that rooted at the node form a tree structure. For example, the tower in Figure 5-2 consists of four trees which is denoted by different colors.

To formulate the problem, we make the following assumptions and notations. We inherit many of them from [41].

- We are given a 3D structure with known geometry. The structure is modeled as a dynamic undirected graph G, whose vertices are points where a robot can grasp if the connecting edges are active. There is a positive cost on each edge.
- Passive bars are reachable from any root nodes.
- Each robot is modeled as two points that corresponds to an anchor and non-anchor gripper on the graph G.
- There are k identical robots on the truss.
- All the robots start at on-truss nodes.
- The robots can sense if an adjacent vertex is free or occupied by another robot.
- Each robot can communicate with the robots that occupy adjacent nodes.
- When two robots communicate, they can share all information (e.g. state, target location, etc.)
- The set of initial pairs of nodes in the graph is R; robot i is initially located at r_i. These locations are not known to the robots.
- The set of target pairs of nodes is T; $T = \{t_1, t_2, ..., t_k\}$
- The goal structure is feasible; given starting locations, there exist a sequence to build it.
- The cost of a set of paths is the sum-total of the edge weights of the paths.
- The goal is for all target nodes to be occupied by the robots and for the overall path cost to be minimal.

The second assumption can be realized when passive bars are supplied at specific locations around *roots* by other robots or machine. For more general case, we will consider that bars are supplied at any position, as future work. Local commnuincation is a reasonable assumption since wide-range communication may spend large amount of energy which is not allowed for a modular robot with a limited power source.

5.3 Distributed Algorithm to Build an Active Structure by Locally Optimal Matching

In this section we propose algorithms for the distributed placement of k robots to build an active structure using distributed locally optimal matching. Our solution is an extension and refinement of 4. In this previous work we considered how to allocate a set of robots to fixed goal locations on a truss in which all the targets are on trusses. Our problem can be defined as finding a matching between R and T, where T can include non-truss nodes.

Each robot runs local algorithms for single robot's locomotion as well as self assembly and disassembly of multi robots. In particular, we focus on how to resolve collisions between robots, because they lead to co-operative reconfigurations and self-assembly.

Each robot's state includes the following data:

- ID: identification number
- Mode: *Single* if alone, *Multi* if in a tree
- Communicating: *true* it is communicating with others
- Status: what it is doing now. *Idle, Busy, Move, Settled, or Assembling*
- Pushing List: a list of the robots pushing it now, to check a cycle
- Location: currently occupying nodes by the anchor and the non-anchor gripper
- Position: only for *MULTI* mode, *ROOT* if it is a root of a tree and *LEAF* otherwise
- Initial and target nodes pair
- Match list: a list of the initial and target nodes pairs it has learned by the collisions only with *settled-down* robots.
- Root-sided robot and Leaf-sided list: a list of which robots are connected to me and how they are connected. There is only one root-sided robot.
- Multi Job: a task for a root robot to do

When a robot is in *Multi* mode in a tree, a root robot of the tree has a role of a local brain to communicate with other robots in the tree and decide what the tree should do.

5.3.1 Algorithm overview

Figure 5-3 shows an expected sequence of building two trees (e.g. columns of a bridge). Each tree is composed of two robots and a passive bar. The following steps are required computationally to carry out this distributed assembly.

- Locate a robot on a truss by locomotion
- Add a unit to a tree
- Cut a unit from a tree
- Control trees to do above tasks

The following sections detail the phases of the algorithm.

5.3.2 Initialization

Using the initial state(mode=*Single*, status=*Idle*), the truss geometry, and the given set of targets, the robot computes the nearest target node, as in the opening part of Algorithm 4.

5.3.3 Deployment

Algorithm4 shows the main control loop. After initialization, each robot executes the distributed deployment algorithm (Algorithm 5 or 6) according to its mode, unless the robot is communicating. Otherwise, it handles messages from the communicating robot. Afterward, it updates its mode and position based on its state (Algorithm 7).

Algorithms 5 and 6 are the procedures that enable a given robot with a target construction tree to move, detect collisions and handle them. The first algorithm checks if this robot has arrived at its desired target. If so, it sets Status=*Settled*, and stops. Otherwise, the robot checks for collisions by communication. If the next node is empty, it takes a

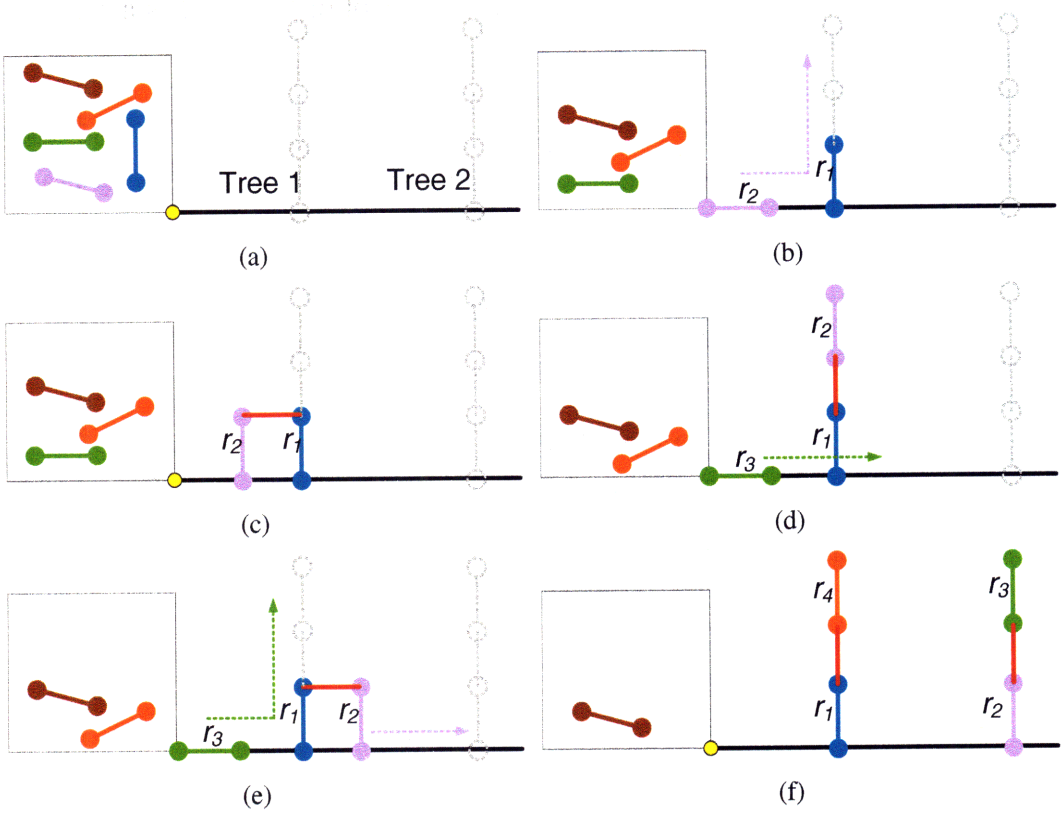


Figure 5-3: Building two trees by four Shady3D-like robots. (a)Robots are in a reservoir with a given design of the columns. The goal structure is denoted by gray dotted lines. In the beginning, every robots goes to the root of the first tree since it is the nearest target. (b)r_1 occupies the root of the first tree. r_2 collides r_1, and finds the next optimal target. (c)r_2 is being added to r_1. (d)r_3 also collides r_1, and finds the next target (the root of the second tree). (e)r_2 goes to the second root instead of r_3, and is being cut from r_1. (f)Sequentially, all the target nodes are occupied by four robots.

Algorithm 4 Main Control

```
1: Initialize state
2: Find the nearest target
3: loop
4:    if Communicating = false then
5:       Distributed Deployment (Algorithm 5, 6)
6:    else
7:       Message handler
8:    end if
9:    Update mode (Algorithm 7)
10:   Wait for Communication in a fixed time
11: end loop
```

step and updates the resources. In case of a collision the robot calls the collision handler (Algorithm 8).

For multi-mode robots, only the *root* robot checks a given job and executes it. Currently, we have only one case for cutting a leaf of the tree, but other behaviors can be added (e.g. locomotion of a tree).

Algorithm 5 Distributed Deployment for *Single* mode

```
1: if reached my targets then
2:    Status=Settled
3:    Add (my start, my target) to match list
4: else if reached one of the target nodes then
5:    target = pair of the current target
6:    Get a new path
7: else
8:    Communicate with adjacent robots
9:    if Next node empty then
10:      Status=Moving
11:      Move to the next node
12:      Update my Location
13:      Swap the anchor
14:      Clear my Pushing list
15:      Status=Idle
16:   else
17:      Status=Busy
18:      Collision handler(collided ID's state) (Algorithm 8)
19:      Status=Idle
20:   end if
21: end if
```

Algorithm 6 Distributed Deployment for *Multi* mode

```
1: switch multi job
2:    case CUTLEAF
3:       cut the leaf (Algorithm 13)
4: end switch
```

Algorithm 7 changes the mode of a robot from *Single* to *Multi* when at least one of its gripping nodes is *not* on a truss and the not-on-truss gripper is on a non-truss node. (r_2 in Figure 5-3(d)) The algorithm decides its position between *Root* and *Leaf*. Mode changes from *Multi* to *Single*, if both the grippers are on trusses, or one is on a truss and other is

not on a truss nor a non-truss node. (r_2 in Figure 5-3(e))) The latter condition is necessary when it is cut from the tree as we will see in Section 5.3.5.

Algorithm 7 Update my mode

```
1: if opposite-anchor gripper is on non-truss node then
2:    mode=Multi
3:    if NO root-sided robot & one of my nodes is on truss then
4:       position=Root
5:    else
6:       position=Leaf
7:    end if
8: else if both my nodes are on truss or (one is on truss & other not on truss nor non-truss
   node) then
9:    mode=Single
10:   clear root-sided and leaf-sided robot lists
11: end if
```

5.3.4 Handling Collisions

The collision handler inherited that in Chapter 4. We introduce new features for self-assembly. Algorithm 8 starts with trying communication to a blocking robot, ID of which is noted as *cID*. SEND(*ID, message*) is a command to transmit the message to a robot that has the ID. This command works only with neighborhood robots. If the collided robot allows communication by sending the message, the handler calls the detailed collision handler. After the detailed handler ends, the algorithm finishes communication by sending the message ENDCOMM, if the robot is still communicating. Note that adding a leaf to a tree may relocate the leaf far away from the root, and the leaf should have ended communication before the addition ends.

Algorithm 9 is the detailed handler. We have found out that the algorithm for *Single* mode (developed in [41],) can be applied for a tree by considering a leaf of the tree as a *Single* mode robot. There are some special cases only for *Multi* mode such as adding and cutting a leaf, which will be explained in Section 5.3.5.

Algorithm 8 Collision Handler(cID)

```
1: SEND(cID,'STARTCOMM')
2: if RecievedMessage≠'LINKED' then
3:    return
4: else
5:    Communicating = true
6:    call handlers(my mode, cID's mode) (Algorithm 9)
7:    if Communicating = true then
8:       SEND(cID,'ENDCOMM')
9:       Communicating = false
10:   end if
11: end if
```

Algorithm 9 Detailed Collision Handler

```
1: if The paths are crossing then
2:    Exchange(my leaf, cID's leaf)
3: else if cID's status≠settled then
4:    Add Pushlist(my PushList + my leaf's ID)
5:    if my leaf's ID ∈ PushList & my leaf's ID = min(PushList) then
6:       Exchange(my leaf, cID's leaf)
7:    else
8:       return
9:    end if
10: else if cID's status=settled & my leaf's target∉ cID's tree then
11:   if my path crosses cID's tree then
12:      SEND(cID,'CUTLEAF')
13:      Exchange(my leaf, cID's leaf)
14:   else if my path goes over cID then
15:      Add me to cID's tree (Algorithm 12)
16:   else
17:      Exchange(my leaf, cID's leaf)
18:   end if
19: else if cID's status=settled & my leaf's target∈ cID's tree then
20:   my target ← a new local optimum
21: end if
```

5.3.5 Multi-robot movement for self-assembly: adding and cutting a leaf

When a robot collides with a settled tree, a leaf of the tree should move instead of the root, and the tree should execute adding and cutting the leaf. We have seen a sequence of handling collision between a robot and a tree in Figure 5-3. When the target of a robot is on the next empty nodes of the tree(Figure 5-3(b),) line 14 of Algorithm 9 is called to add the robot to the tree. The robot is added to the tree, connected by the red passive bar (Figure 5-3(c)). When the path of a robot is crossing the tree (Figure 5-3(d)). The robot requests to cut a leaf from the tree according to line 11 of Algorithm 9, and it exchanges its identity with the leaf. At the next turn of the main control loop, it has the target of the leaf, while the leaf is being cut (Figure 5-3(e)).

To implement these behaviors, communication along a tree is required. We call it SENDTREE as in Algorithm 10, where a simple depth-first search algorithm is used with the state of leaf-sided robot list and root-sided robot. A robot in the tree propagates a received message from its root-sided robot to leaf-sided ones, waits for the answer of the leaf-sided, and finish communication with the root-sided.

Algorithm 10 SENDTREE: communication in tree

```
1: for leaf-sided robot list do
2:     SENDTREE(received message)
3:     do my job
4:     WAIT('DONE')
5: end for
6: SEND(root-sided robot, 'ENDCOMM')
```

Add a leaf

When a tree adds a leaf, the robot uses Algorithm 11, while the new leaf works with Algorithm 12 in parallel. When the new leaf sends the message 'ADDLEAF', the root checks if a bar is connected to the new leaf's target node. If the bar does not exist, the tree makes a path to a new bar by inverse kinematics , which will be explained in Section 5.4. The path consists of new locations of the nodes in the tree, from the root to a side of the bar.

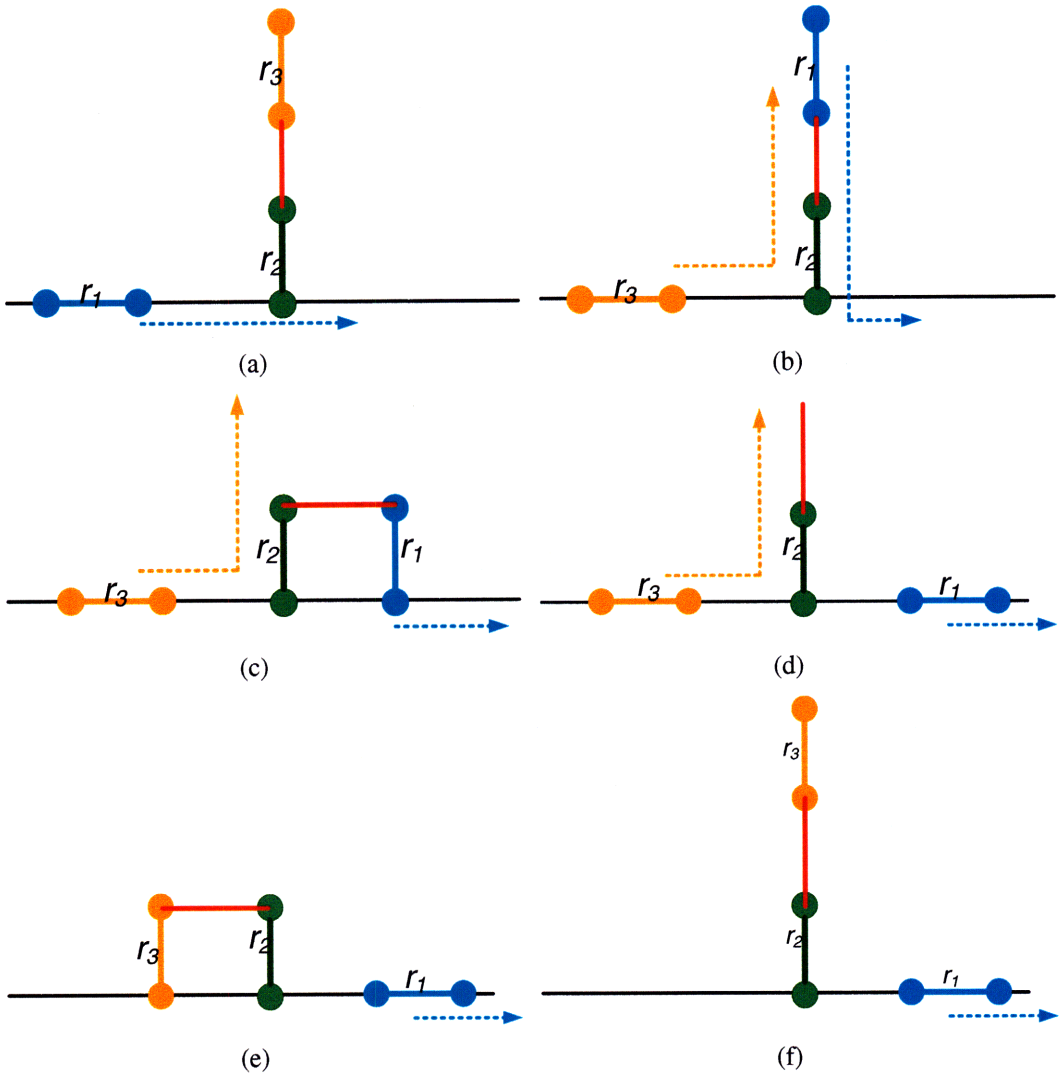


Figure 5-4: A sequence of collision handling between a robot and a tree. (a)r_1 is crossing a tree rooted by r_2. (b)The tree finds the nearest leaf r_3 to cut and exchange it with r_1. (c)The root is cutting r_1 from the tree. (d)The tree returns to the original location. (e)r_3 is being added to r_2. (f)The tree is re-constructed with the same robots, while r_1 has crossed the tree.

The root robot sends the path to the others in the tree, and robots that are on the path move with the given locations. The root commands to grasp the bar, and calculates another path to the anchor node of the new leaf. Afterward, the root sends the corresponding locations to the new leaf, while the tree moves with the new path. They exchange some messages to synchronize grasping and releasing, and end communication. Note that communication finishes before the addition is completed. Otherwise the leaf may move far away from the communication range. Finally, the tree returns to its original locations and updates the match list and the graph.

Algorithm 11 Add a new leaf to my tree

```
1: Status=Assembling
2: Check a bar at new leaf's target (SENDTREE)
3: if bar not exists then
4:     Get path to bar by the inverse kinematics
5:     Move my tree (SENDTREE)
6:     Grasp the bar (SENDTREE)
7: end if
8: Get path to new leaf's target by inverse kinematics
9: Move my tree (SENDTREE)
10: SEND(new leaf's ID,'COORDINATE')
11: WAIT(new leaf's ID,'READYASSEMBLE')
12: SEND(new leaf's ID,'ASSEMBLE')
13: end communication with new leaf
14: Return to original locations (SENDTREE)
15: Update match lists and map (SENDTREE)
```

Algorithm 12 Add myself to tree

```
1: Status=Assembling
2: SEND(cID,'ADDLEAF')
3: move to Received Message's new Location
4: SEND(cID,'READYASSEMBLE')
5: Grasp a bar
6: SEND(cID,'ASSEMBLED')
7: Merge match lists
8: Status=Settled
```

Cut a leaf

Adding a leaf happens between colliding robots. Cutting a leaf is a procedure between robots in a tree, and is designed as an independent process. Cutting a leaf is called when a root robot has state MultiJob=*CUTLEAF* (Algorithm 6), which is triggered by the colliding robot (Algorithm 9). Algorithm 13 shows how cutting a leaf is implemented at the root. The root begins to cut a leaf as it finds a next node on a truss where the leaf should be located. If the node is empty, the root gets a path to it by inverse kinematics, reconfigures the tree, grasps the node, and release the leaf from the tree. Otherwise the collision handler is called.

Algorithm 13 Cut leaf

```
1: Find next node for the leaf
2: Communicate with adjacent robots
3: if next node is empty then
4:     Get path to the next node
5:     Move my tree (SENDTREE)
6:     Grasp the next node (SENDTREE)
7:     Cut leaf from tree and update map (SENDTREE)
8:     Return to original locations (SENDTREE)
9:     Multi Job = false
10: else
11:    Collision handler
12: end if
```

5.4 Controlling linkages by inverse kinematics

In this section, we introduce inverse kinematics for a tree with multi-robots. This controls the partial linkages to execute the necessary movements for adding or cutting a leaf. The solution works in near-singular regions as well as is specially designed for our active structure.

5.4.1 Approximated solution for multi-robots

Reaching an arbitrary point in space by linked robots requires robot coordination. Unfortunately, the structure of Shady3D does not allow a closed-form inverse kinematics solution

even for the simplest 6DOF linkages from two robots. Instead of using an explicit solution, we use an approximation algorithm based on the manipulator jacobian. We select a Damped Least Square (DLS) method because it has good robustness and performance [5].

The equations for joint angles are:

$$\Delta\theta = J^T(JJ^T + \lambda^2 I)^{-1}\Delta p,$$

where J is the manipulator jacobian and λ is a constant that we have to tune. Our implementation for Shady3D has the following procedure:

- Get a target displacement from the current configuration
- *Clamping*: divide the displacement into small pieces enough that the jacobian approximation is valid.
- For each divided displacement, get angle displacements by the DLS
- Update the current configuration by adding the joint displacements

These procedures work well with almost no error, when the target posture is away from any singularities. For the self-assembled 6DOF arm, singularities occur when two robots are fully stretched out, or when all four gripper points are located in the same plane. With a general DLS, lifting up the end-effector along the vertical(Z) axis gives arise of unwanted deviation along Y-axis and Z-axis. Unfortunately, it is theoretically impossible to extract an exact solution from jacobian at singularities. However, we can establish a trade-off: usually position error is more critical than orientation error. To compensate for the big position error, we propose a modified clamping method: DLS with *Variable Clamping Constant* in which we clamp position and orientation separately as follows:

$$\Delta\hat{x} = \begin{cases} \Delta x & \text{for} \quad \|\Delta x\| \le c_x \\ c_x \frac{\Delta x}{\|\Delta x\|} & \text{for} \quad \|\Delta x\| > c_x \end{cases}$$

$$\Delta\hat{\varphi} = \begin{cases} \Delta\varphi & \text{for} \quad \|\Delta\varphi\| \le c_\varphi \\ c_\varphi \frac{\Delta\varphi}{\|\Delta\varphi\|} & \text{for} \quad \|\Delta\varphi\| > c_\varphi \end{cases},$$

where Δx and $\Delta\varphi$ are the clamped position and orientation, and c_x and c_φ are clamping constants, respectively. We also use a larger clamping constant nearby singularities so that we get less errors. The final clamping constants are:

$$c = \begin{cases} c_{max} & det(J) \geq J_0 \\ c_{min} + \left|\frac{det(J)}{J_0}\right|^2 (c_{max} - c_{min}) + k\frac{\Delta p}{\|\Delta p\|} & det(J) < J_0 \end{cases},$$

where J_0 and k are tunable constants, and Δp is the clamped displacement. The last term of the lower c is added to accelerate the convergence.

We have observed that the proposed method yields only 2mm position error in a case of lifting up(100mm) of 6DOF linkages from the singular posture while the original DLS gives 23mm error, by compromising with a larger orientation error (0.5 to 3.5 degree).

5.4.2 Node-based inverse kinematics

Another difficulty lies in that our Shady3Ds with various configurations can result in the same tree structure; it is hard to directly get the right joint angles as well as to configure a tree with input joint angles. We use the node-based inverse kinematics so that its outputs are new locations of the input nodes rather than joint angles. Since we have a closed-form inverse kinematics solution to connect two nodes by a robot (a robot is using this whenever it locomotes on a truss), we do not need to consider combinations of each robot's configuration. The following procedures are implemented:

- Get a path - composed of nodes - to the leaf to move.
- Assume the simplest joint configuration to match the path. We can use the inverse kinematics for a single robot for every two nodes.
- Calculate new joint angles by the inverse kinematics in 5.4.1.
- Get new locations of the nodes from the forward kinematics of the tree.

The output path includes indexes and new locations of the nodes in a tree. When a root wants to move a tree, it sends the path to its leaves by the SENDTREE protocol. The

receiving robots can have the corresponding new location, and they solve the single-robot inverse kinematics to reconfigure themselves.

5.5 Analysis

In this section, we briefly review the previous analysis of the optimality for distributed matching [41]; the distributed algorithm was shown to have $O(k^2)$ asymptotic competitive ratio to the global optimum, wheares a greeday algorithm which seeks the nearest next target has an exponential competitive ratio. We prove that the same bound still holds for our dynamic graph.

5.5.1 Optimality of the distributed matching for a static graph

Our analysis inherited all the results on minimal weight partial matching in Chapter 4.

5.5.2 Optimality for the dynamic graph

The key idea for proving the competitive ratio of the distributed matching for dynamic graphs is that a non-truss node should have its root on the truss. This implies that there must exist a closer root node from a robot than any non-truss nodes.

Lemma 10 *The locally optimal matching only includes connected target nodes in a given dynamic graph. i.e. it never has target nodes that do not have any activated edges.*

Proof: *We use induction. When a robot starts, the proof is trivial. Suppose the lemma holds for $k-1$ robots and it finds a new target t_k, not connected in its graph, with the locally optimal matching M_k. Let us say r_k is matched to t_k in M_k. Note that we can always find a connected target node $\hat{t}_k$, in a tree that includes t_k, which are closer to a truss than t_k, because t_k should be connected to a truss by a tree with $\hat{t}_k$. Now we have the better matching $\hat{M}_k$ by coupling r_k to $\hat{t}_k$ with maintaining the other matching in M_k. It is contradiction. Therefore the lemma holds for k robots.*

Lemma 11 *The distributed matching algorithm for our dynamic graph has the same competitive ratio $O(k^2)$ as that for a static graph.*

Proof: *By Lemma 10, the locally optimal matching will find a new target that should be connected in the given graph, no matter the algorithm uses a fully connected graph or the given one. Therefore, the algorithm has the same competitive ratio as that uses a static graph as in [41].*

5.6 Implementation

We have implemented the distributed placement algorithm to build an active structure in Java. We simulated each robot as an independent process (thread) to ensure parallelism.

Figure 5-5 shows snapshots of building a hand on H-structured trusses. 18 robots are deployed and passive bars are around the root nodes. Yellow and green circles denote each gripper of a robot, and pink bars are passive bar. Small red circles are target nodes which compose five trees. Robots start from side trusses and gather into the center as they perform successive add-leaf and cut-leaf operations. For a better view, passive bars appear only when they are grasped. 13 bars are used to connect the robots.

Figure 5-6 is implementation of building a tower we suggested in 5.1. The tower consists of four trees, each of which has 3 robots and 2 bars. Note that the implementation of the tower in 5.1 was done by a central controller that knows the exact sequence of motions generated by hands.

The brief statistical summary of the simulations is in Table 5.1. Collision among trees is ignored, and will be considered in our future work.

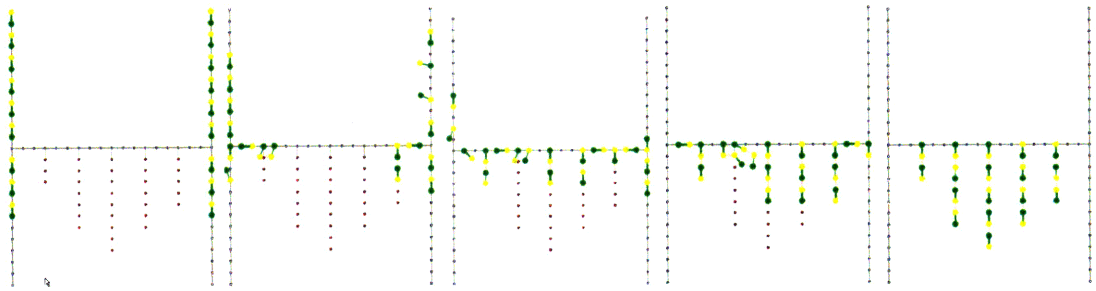


Figure 5-5: Snapshot of building a hand-like active structure. Thick gray lines are trusses, while thin ones are edges to connect non-truss nodes. 18 robots and 13 passive bars are connected. Yellow circles are the left grippers and greens are the right ones. The bars are pink.

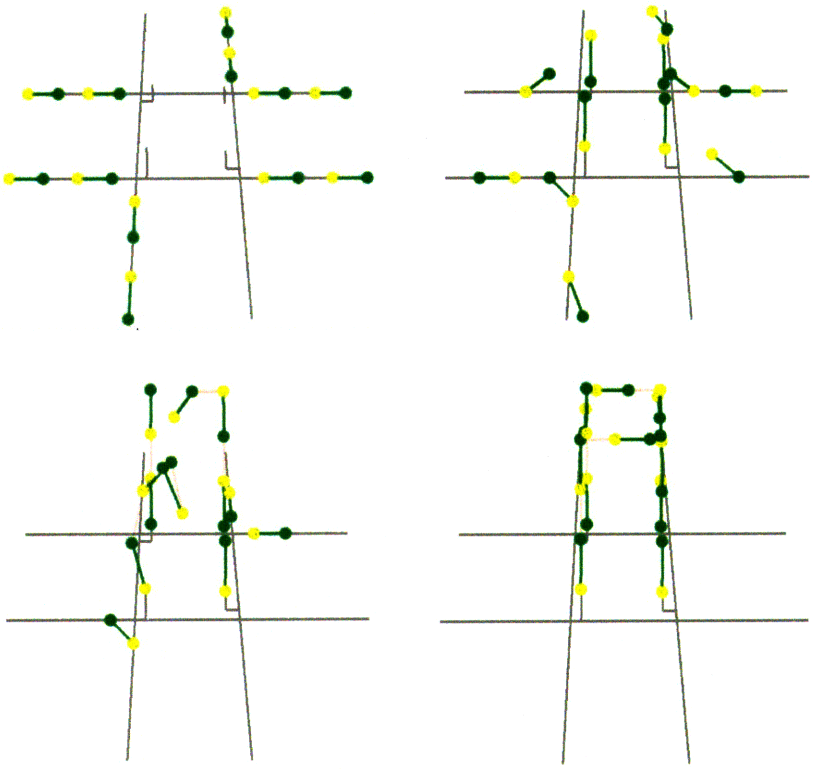


Figure 5-6: Snapshot of building a tower we proposed in 5.1. 12 robots and 8 passive bars are used.

Table 5.1: Count of the operations

Count of operations / Structure	Hand	Tower
Total move/Optimum	297/262	84/66
Average communication	40	8.3
Exchange	70	6
Getting a new optimum	36	18
Adding a leaf	34	10
Cutting a leaf	22	2

Chapter 6

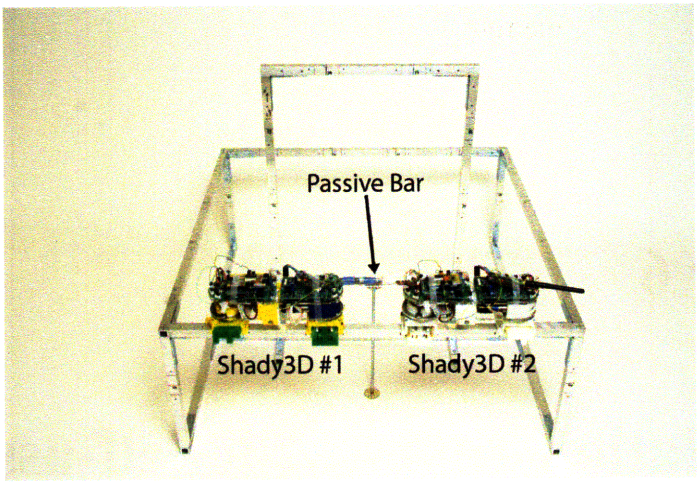
Manipulation Tasks with Self-assembled Arm

6.1 Self-assembly of two Shady3Ds

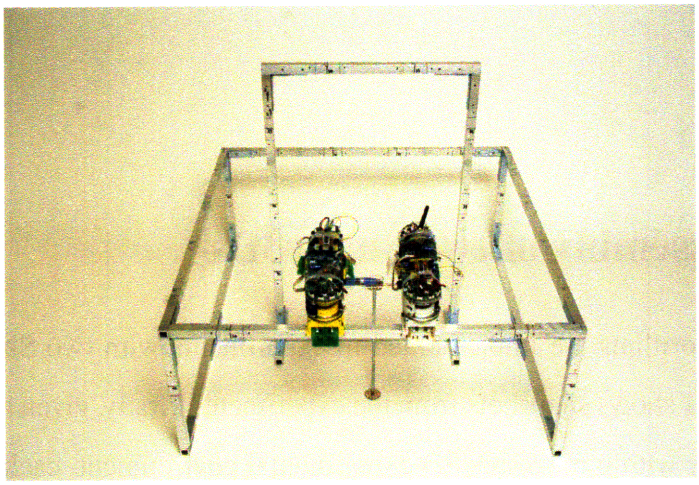
The proposed algorithms are implemented in experiments with two Shady3D robots and one bar. Figure 6-1 shows snapshots from the experiment. Firstly, given a specified position for the passive bar within the Shady3D experimental environment, each Shady3D module optimally positions itself so as to be able to reach the bar. Details of the optimal deploying algorithm are addressed in [41]. In the first step of the algorithm each Shady3D module moves independently and in parallel to reach and grasp the bar. The bar is detected using the LED sensors within the Shady3D grippers. Upon grasping the bar, the Shady3D modules signal to each other using Bluetooth to coordinate the completion of the grasping step and the self-assembly of a 6DOF manipulator. We tested the self-assembly, and a sequence of 10 executions resulted in no error. Each self-assembly experiment took 1 minute (See Table 4.2).

6.2 Task Execution

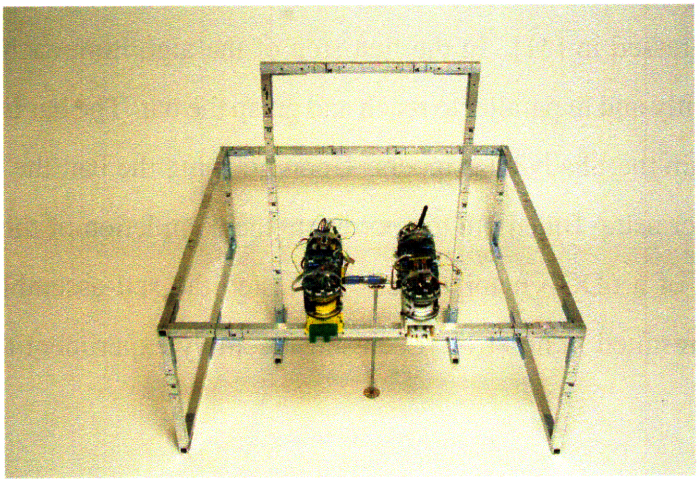
We have developed algorithms for four kinds of tasks with the manipulator. The algorithms were implemented on our physical prototype 6DOF modular manipulator. In each case,



(a)



(b)



(c)

Figure 6-1: Implementation of self-assembly of 6DOF modular arm (a) Two robots have moved to the approachable nodes. (b) They are swinging their body to find the bar. (c) They have grasped each side of the bar.

task information is given to the robots in the form of a command stack. The robots decide which role to play based on the task specification and its location.

6.3 Distributed control algorithm for task execution

Each task is a stack of command sets for the two robots, and how a robot execute the task is shown in Algorithm 14. Parameters of the command set are:

- *RootNode (#)*: the root location to anchor the arm
- *Displacement (x,y,z,roll,pitch,yaw / $\theta_1 \cdots \theta_6$)*: 6 joint movements and end-effector displacement
- *Grasp (G/R)*: grasp/release of the end-effector

Each robot starts by finding out if it is a root. The root robot calculates the joint displacement of two robots directly or indirectly by inverse kinematics. The leaf robot waits for a command. The root sends the corresponding joint displacements to the leaf robot. Then they both execute their next command in parallel. The root checks the command completion, and then pops the next command set until the stack is empty.

6.4 XYZ-directional movement

In this task, the distributed inverse kinematics protocol is used to implement the positioning of the arm's end effector at a desired location (x, y, z). The arm's initial configuration is shown in Figure 6-2(a). The left gripper of the arm is the anchor and the right gripper is the end-effector. We have tested different (x, y, z) locations for the 6DOF manipulator built in Section 6.1 as shown in Figure 6-2(b-c). Each experiment was done 10 times without error and it took 20 seconds(See Table 6.1.) In this case, the task stack has only one command set with a single end-effector displacement.

One challenge is coping with the position error along the vertical axis - in this case, Z-directional - because of tilting of the arm due to gravity. About 20mm error was measured

Algorithm 14 Task execution

```
1: while Task Stack not empty do
2:   Pop the next queue
3:   if Anchor = Root then
4:     Get the commands from the queue
5:     Send the command for the leaf
6:     State = Moving
7:     Execute my command
8:     while The leaf's State = Moving do
9:       Delay
10:    end while
11:    State = Assembled
12:  else
13:    Wait for the command from the root
14:    State = Moving
15:    Execute my command
16:    State = Assembled
17:  end if
18: end while
```

regardless of the Z-directional displacement. The error mainly comes from mechanical weakness of a robot (e.g. backlash, tolerances, and plastic material).

6.5 Reaching nodes unreachable by one robot

Consider an inspection task which requires reaching every point on the truss. As pointed out in [45], some points on the truss are unreachable by one robot due to its fixed length and 3DOF. When we model the truss environment as a graph where nodes are points of interest and edges correspond to reachability among the nodes, such unreachable points are nodes without an edge. Upon self-assembly, many unreachable points become reachable by the 6DOF linkage because of enhanced workspace and additional DOFs.

Figure 6-3 shows the self-assembled robot built in Section 6.1. reaching the unreachable nodes(denoted by the arrows). The task stack has one command set with a single end effector displacement according to 3-D locations of the nodes. Three unreachable nodes were tested ten times each without error. Each task took 40 seconds(See Table 6.1.) The position error along the vertical axis due to the mechanical weakness of the arm persists

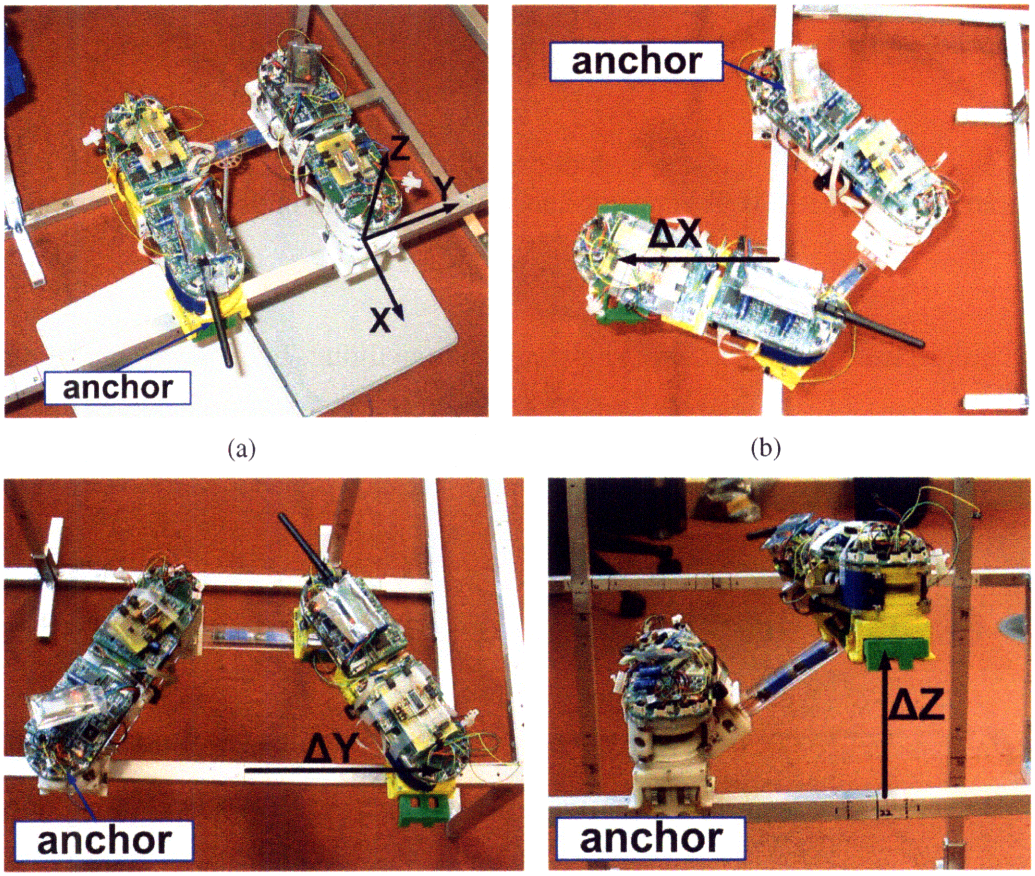
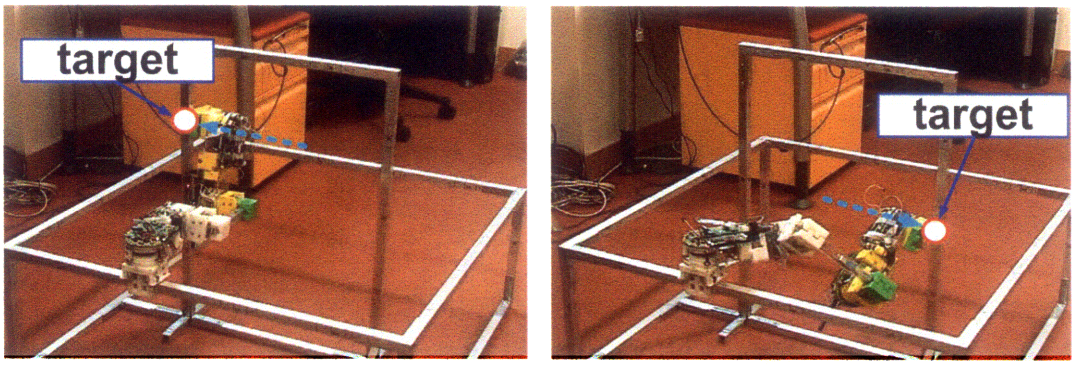


Figure 6-2: Uni-directional movement of a 6DOF manipulator composed of self-assembled Shady3Ds. (a) A self-assembled manipulator with two Shady3Ds. The left gripper is anchored at the truss and the right one is free to move. (b) X-directional movement with 150mm displacement (c) Y-directional movement with 150mm displacement (d) Z-directional movement with 150mm displacement



(a) (b)

Figure 6-3: A 6DOF manipulator with two Shady3Ds reaches some nodes which are unreachable by one robot. The robot can be anchored anywhere in the environment.

for the task as well with an observed maximum 60mm tilting. In our environment, the self assembled 6DOF can reach all the nodes.

6.6 Pick and drop by forward kinematic control

In this task, the arm collects an object(a bar), moves to a different location where it drops the object. This task requires a 6DOF manipulator. The locations of pick and drop are given by joint angles. The robot moves by distributed forward kinematic control.

The task stack is composed of 7 command sets each of which has one joint displacement or grasping/release. As the task starts, one of the modules releases its grasp of the environment. Figures 6-4(a, b, c) shows two modules controlled independently and in parallel to demonstrate the movement of the arm. An additional bar is manually presented to the free gripper of the 6DOF manipulator. The bar is grasped, transported, and dropped at a specified location (see Figure 6-4 (d, e, f).) We have performed this experiment 10 times in a row during the course of one hour. Each experiment consisted of 9 joint movements and 5 grasping/release operations, and it took about 140 seconds. All the control steps succeeded for all the experiments. However, due to a hardware failure at the end of the 7th experiment one of the gripper motors had to be replaced(See Table 6.1.)

6.7 Locomotion

In this task we demonstrate that the modular arm is mobile. The previous tasks have a fixed anchor point. Locomotion of the arm allows arbitrary anchor points.

Figure 6-5 shows snapshots of locomotion on truss segment. The arm moves by alternating the left and right anchors and inching along. The task stack has 3 command sets. The gripper located in opposite to the direction of locomotion(left) is set as the anchor. After the right gripper moves, the robot swaps the anchor gripper, and the left gripper moves.

6.8 Discussion

A summary of our experiments is shown in Table 6.1. The biggest problem is caused by the structure's tilting error due to gravity. This is a problem with the experimental device not the algorithm. Within a unit alone, this problem is small and can be compensated [45]. However, the self-assembled robot is three times longer than an individual module, which causes a big moment and tilting. In the near term, we will reduce the error by better hardware as well as a compensating algorithm.

Table 6.1: Result of the Experiments

experiment	number of execution	number of joint displacement	number of grasping/release	success ratio	operation time(sec)	remark (error)
Self assembly	10	6	4	10/10	60	
Pick and drop	10	5	3	9/10	140	motor failure
XYZ move	30	6	0	30/30	20	tilting error
Reaching	30	6	0	30/30	40	tilting error

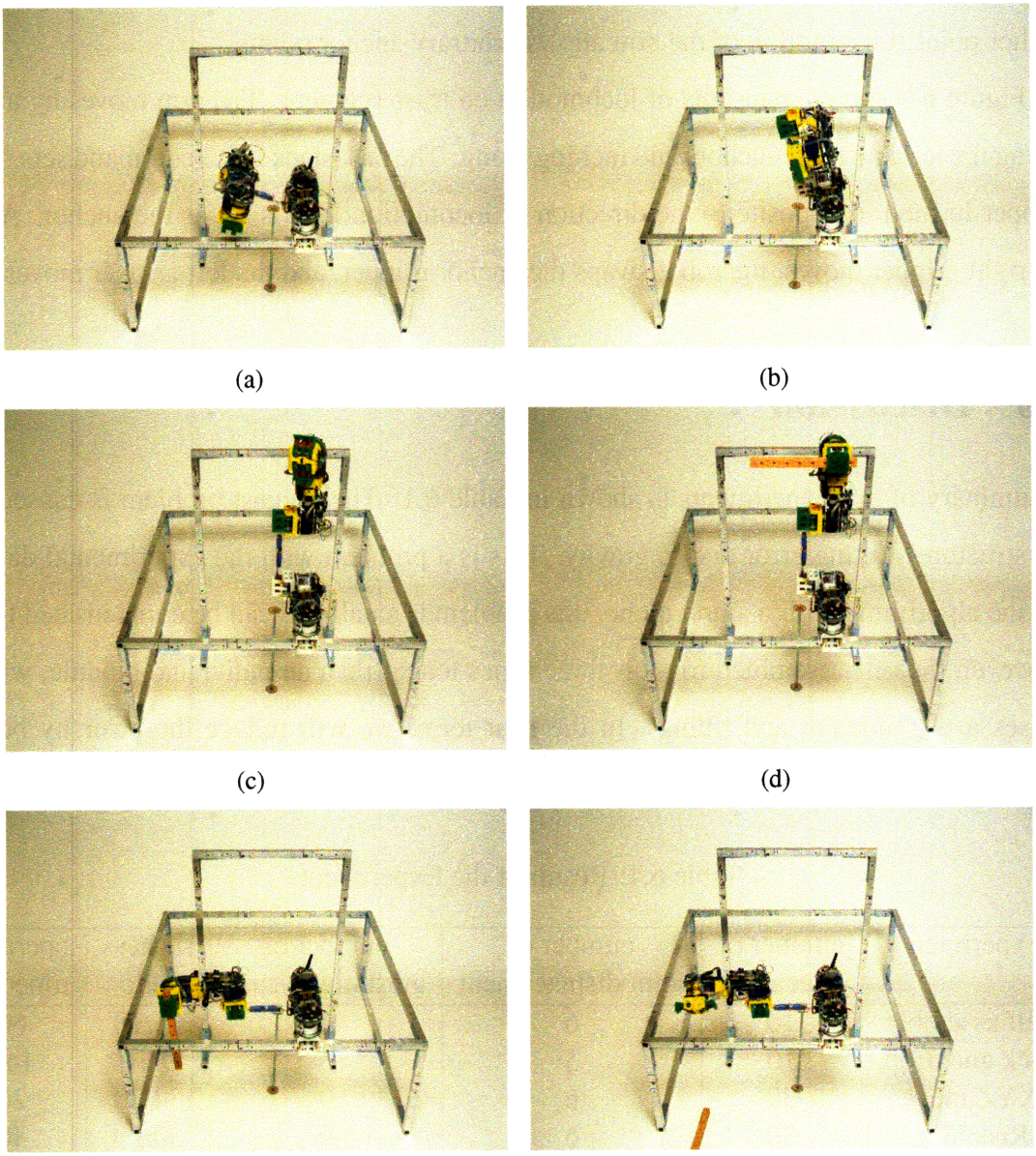
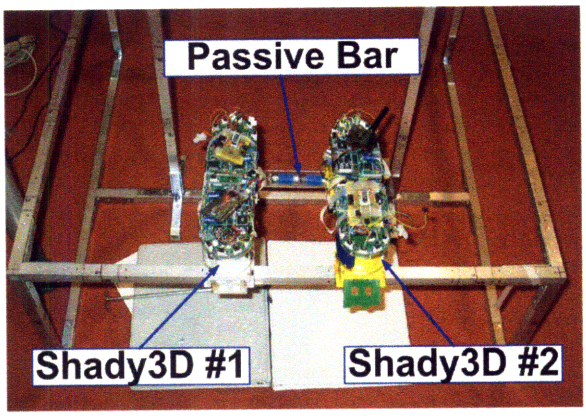
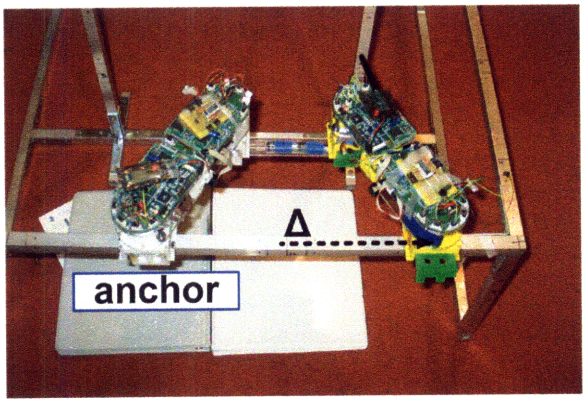


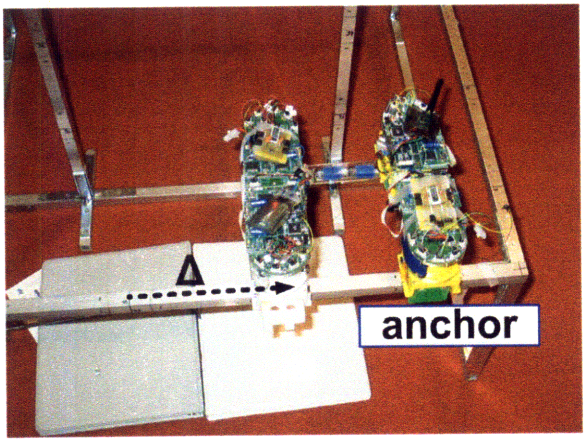
Figure 6-4: Implementation of moving a bar. (a) A 6DOF manipulator combined by two modules and the base module pulls the other upward. (b) The base module has fully moved the other module up. (c) The manipulator is stretched to the maximum height. (d) The end effector is given a bar to be moved. (e) The bar is moved to the dropping position. (f) The manipulator has dropped the bar.



(a)



(b)



(c)

Figure 6-5: Locomotion of a self-assembled manipulator (a) After self-assembly, a 6DOF manipulator releases the right gripper. (b) The right gripper moves to the next right node. (c) It exchanges the anchor and the left gripper moves to the right node.

Chapter 7

Conclusion and Future work

This thesis describes algorithms and implementation of building a self-assembled robot composed of passive components and modular manipulators. Reconfiguration of the self-assembled structure is also introduced. We developed this work in the context of a modular mobile manipulator Shady3D. We designed a module with the minimal number of joints for 3D movement and building a 6DOF manipulator. By combining two modules and one passive bar, we can generate a more capable robot. We see self-assembly as an extension of truss navigation of the truss climbing robots. We aimed a distributed and an optimal solution.

For truss navigation, we proposed a distributed localized algorithm for placing multiple identical robots at desired locations on a truss in a path-optimal way. The algorithm is based on successive locally optimal matching , and results in a perfect min-cost bipartite matching between the initial nodes and the target nodes. The robots discover the target nodes incrementally through collisions and self-organize as the solution. Collisions are handled according to five different cases. Our solution is feasible, and has a quadratic competitive ratio $O(k^2)$ which is much more efficient than the greedy solution which exhibits exponential competitive ratio. We have also analyzed that running time as $O(k^5 + k^2(n+m)logn)$ as compared to the centralized offline algorithm with $O(k(k^2+n+m)log(n+k^2))$ runtime. In the simulations, we have found out the algorithm works very efficiently in a view of the cost and the communication. Finally, we applied it to the real system with Shady3Ds and additional inert robot modules.

Extending the truss navigation, we developed a unified approach to implement building an active structure by self-assembly using a dynamic graph and distributed matching. We developed this work in the context of a modular mobile manipulator Shady3D. The algorithm works in a distributed way so that robots depend on only local information. A target structure is modeled as a dynamic graph with edges that are not activated until one of the incident nodes is occupied by a robot. The robots discover their locations in the structure incrementally through collisions, and update their graph. Self-assembly of the robots makes a tree with robots and bars, and a root of the tree becomes a local brain to control the leaves with the node-based inverse kinematics. Adding and cutting a leaf are implemented by communication between a robot to be added or cut and the root of the tree. The robots build the structure in a locally optimal way, and we have proved that the same competitive ratio for a static graph holds for our dynamic graph.

Finally, we described a suite of algorithms and experiments for reconfiguring the self-assembled structure. Hardware implementation of building a 6DOF manipulator and several tasks show how the proposed self-assembly works in the real world. The coordinated manipulation algorithms perform well. They are generally robust and the response time is adequate for the tasks we considered. However, the materials used in the prototype cause a structured tilting error which has to be eliminated in future versions.

7.1 Lesson learned

Since we use a new self-assembling system, we have experienced many difficulties and also learned plenty of lessons. The belows are the lessons we have obtained.

Algorithm The distributed matching algorithm works perfectly with the selection criteria. We proved a quadratic competitive ratio to the global optimum, however, we could not prove the tight bound for the distributed matching. We believe the actual competitive ratio of our algorithm is *linear* since we do not have any counter examples.

We have not focused on the optimal resource altough amount of the resource via communication is important in reality. Our assumption is that a robot see all the resource of

the communicating robot, however, that may cause a serious lag in communication when many robots gather. What is the best set of infomation for self-assembly has not revealed yet, and we need to think over.

Hardware The passive bar we designed works fine in a very short range as in the experiments of two Shady3Ds. If we extend the experiments to more robots in 3D space, finding a location of the bar may become a serious problem because it requires calibration of 6-DOF. Also, currently communication between the robot and the bar is simply transmitting on/off information. For more objects for the robot to handle, the bar should be *smarter*. For example, the bar need to talk to the robot about its infomation and current state, etc.

Implementation The most difficult point of the experimental implementation was tilting from gravity. Even with two Shady3Ds, we had big tilt that caused most of failures in the locomotion experiments.

Besides, now Shady3D can lift up only one modules and a 3 link manipulator with 3 Shady3Ds is not implementable on earth. We may need to introduce a parallel mechanism to support a longer structure.

Appendix A

Passive Bar specifications

Figure A-1 shows a CAD model of the passive bar. The size of the bar was chosen to fit the size of the gripper of Shady3D. The distance between the two holes for the IR LED is the same as gripper to gripper distance of Shady3D.

The Schematic for the IR LEDs is shown in Figure A-2.

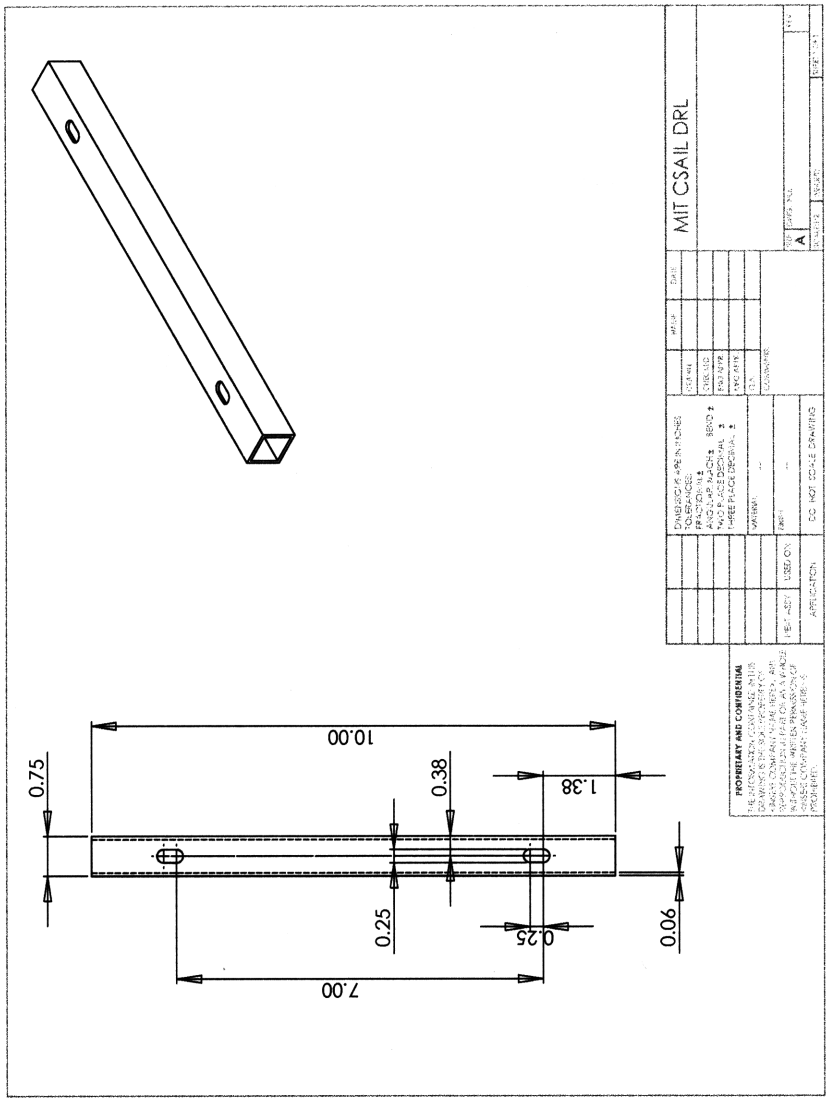


Figure A-1: CAD model of the passive bar. The IR LEDs are located in the holes.

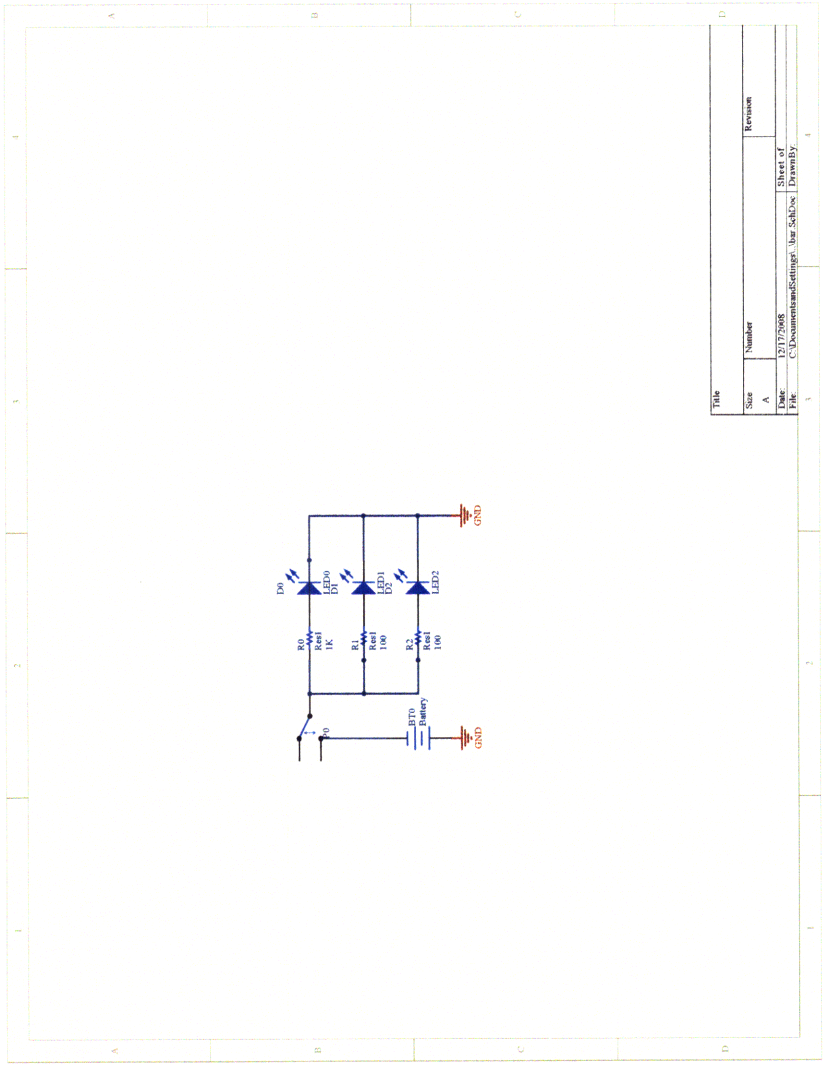


Figure A-2: Schematic for the passive bar. A LED is for checking the power on/off. Two IR LEDs turn on/off by the switch.

Bibliography

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

[2] A. Wolf, H. B. Brown, R. Casciola, A. Costa, M. werin, E. Shamas, and H. Choset. A mobile hyper redundant mechanism for search and rescue tasks. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2889–2895, Las Vegas, Nevada, October 2003.

[3] Aaron Greenfield, Alfred A. Rizzi, and Howie Choset. Dynamic ambiguities in frictional rigid-body systems with application to climbing via bracing. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1959–1964, Barcelona, Spain, April 2005.

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

[5] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose pseudoinverse and damped least squares methods. April 2004.

[6] Zack Butler and Daniela Rus. Distributed planning and control for modular robots with unit-compressible modules. *International Journal of Robotics Research*, 22:699–715, 2003.

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

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

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

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

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

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

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

[14] Carrick Detweiler, Marsette Vona, Keith Kotay, and Daniela Rus. Hierarchical control for self-assembling mobile trusses with passive and active links. In *IEEE Intl. Conf. on Robotics and Automation*, pages 1483–1490, 2006.

[15] J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM*, 19:248–264, 1972.

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

[17] Gregory S. Chirikjian. General methods for computing hyper-redundant manipulator inverse kinematics. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1067–1073, Yokohama, Japan, 1993.

[18] Gregory S. Chirikjian and Joel W. Burdick. A hyper-redundant manipulator. *IEEE Robotics & Automation Magazine*, pages 22–29, December 1994.

[19] Gregory S. Chirikjian and Joel W. Burdick. The kinematics of hyper-redundant robot locomotion, 1995.

[20] Gregory S. Chirikjian, Yu Zhou, and Jackrit Suthakorn. Self-replicating robots for lunar development. *IEEE/ASME Transactions on Mechatronics*, 7(4):462–472, 2002.

[21] H. W. Hamacher and S. A. Tjandra. Mathematical modeling of evacuation problems: State of the art. pedestrian and evacuation dynamics. Technical report, Institut Techno- und Wirtschaftsmathematik.

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

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

[24] Jackrit Suthakorn and Gregory S. Chirikjian. A new inverse kinematics algorithm for binary manipulators with many actuators. *Advanced Robotics*, 15(2):225–244, 2001.

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

[26] B. Kalayanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3), 1993.

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

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

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

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

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

[32] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.

[33] W.H. Lee and A. Sanderson. Dynamic analysis and distributed control of the tetrabot modular reconfigurable robot system. *Autonomous Robots*, 10(1):67–82, 2001.

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

[35] A. Meyerson, A. Nanavati, and L. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.

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

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

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

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

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

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

[42] Seung-kook Yun and Daniela Rus. Self assembly of modular manipulators with active and passive modules. In *Proc. of IEEE/RSJ IEEE International Conference on Robotics and Automation (to appear)*, Pasadena, CA, May 2008.

[43] Tanio K. Tanev. Kinematics of a hybrid (parallel-serial) robot manipulator. *Mechanism and Machine Theory*, 35:1183–1196, 2000.

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

[45] Yeoreum Yoon. Modular robots for making and climbing 3-d trusses. Master's thesis, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, 2006.

[46] Yeoreum Yoon and Daniela Rus. Shady3d: A robot that climbs 3d trusses. In *Proceedings of the International Conference of Robotics and Automation*, 2007.

[47] Yeoreum Yun and Daniela Rus. Shady3d:a robot that climbs 3d trusses. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4071–4076.

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