Distributed Auction-Based Initialization of Mobile Robot Formations

by Rob Long, Bachelor of Science

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Master of Science Degree

Department of Computer Science
in the Graduate School
Southern Illinois University Edwardsville
Edwardsville, Illinois

August, 2010

ABSTRACT

DISTRIBUTED AUCTION BASED INITIALIZATION
OF MOBILE ROBOT FORMATIONS

by

Rob Long

Chairperson: Dr. Jerry B. Weinberg

The field of multi-robot coordination, specifically robot formation control, is rapidly expanding, with many applications: reconnaissance, urban search and rescue, surveying, sensor networks, and exploration. One of the most compelling applications considered is that of space-based solar power collection, in which, satellites are placed in outer-space in order to harvest sunlight directly, before it is filtered by the earth's atmosphere. As Bekey et al [5] have suggested, an excellent solution to the problem of building such solar collectors is that of multi-robot formations. Mead et al [19] have begun the work of describing a method to fully address this concept. This project addresses an issue raised in Mead's work: how to initialize the formation of robots from an unorganized swarm.

This work explores two distributed auction-based methods to autonomously initialize and reorganize the network structure of a formation of mobile robots. The *push auction method* casts the members of the formation as auctioneers and unassigned robots as bidders on neighboring positions within the formation. Unassigned robots become members of the formation as they win auctions and get pushed onto the endpoints of the formation. The *insertion auction method* reverses the roles and casts the unassigned robots as auctioneers and the members of the formation as bidders with unassigned robots being inserted into the formation in the position of the winning bidder.

The two methods were implemented in simulation. Experiments varying the size and shape of the formation were conducted on both methods. The results were evaluated with regards to several parameters: time to converge, average time for a robot to join the formation, total distance traveled, distance traveled per agent, total messages sent, messages sent per agent, and messages sent per time step.

ACKNOWLEDGEMENTS

Be it acknowledged that I wish to thank the following:

- Dr. Jerry B. Weinberg, an excellent computer scientist, a gifted teacher, and a wily administrator. Your inexhaustible ability to read and re-read my run-on sentences and sleep-deprived gibbering is remarkable.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION AND SIGNIFICANCE

Problem Statement

The field of mobile robotics, particularly formations of and interactions among mobile robots, is a growing area of study. Proposed applications for large formations of mobile robots are numerous already and include many domains: search and rescue, battlefield reconnaissance, exploration and survey, and adaptive structures. Some proposals call for formations of robotic satellites, specifically space-based solar power collection [5] and sparse aperture telescopes like NASA's Terrestrial Planet Finder [2] and the ESA's Darwin [9; Fig. 1].

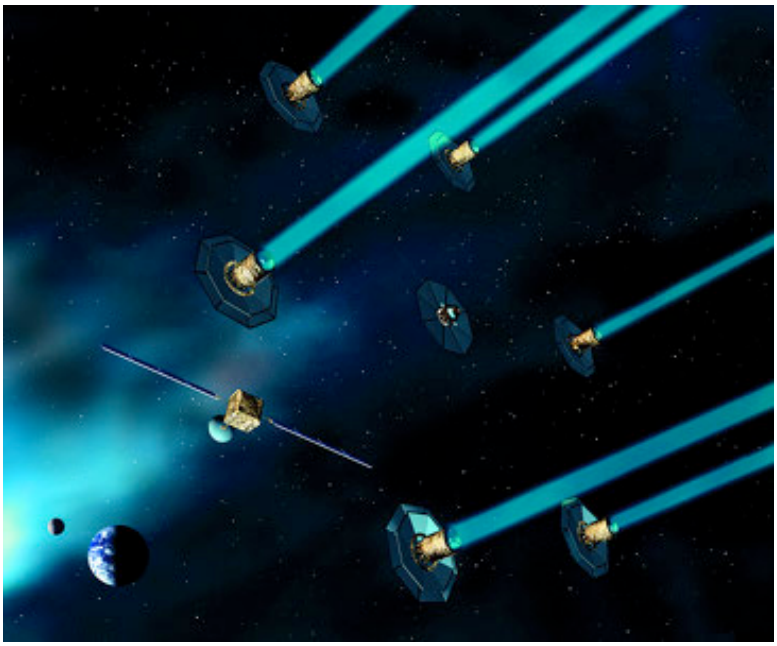


Fig.1: An Illustration of ESA's Darwin [9]

Such applications call for large amounts of homogeneous mobile robotic satellites to maintain a fixed formation, maneuver and maintain formation, and exhibit fault-tolerance such that the total disabling of an individual agent can occur without affecting the function of system as a whole.

Formation Control

Solutions to the problem of command and control of a formation of robots fall into two main categories. The first approach, a top down, hierarchical approach, can be implemented as either a system consisting only of mobile robots, where one robot is assigned as a leader and issues orders to the other members of the formation, or as a group of robots controlled by a central planner with a view of all the robots, which then issues commands to control all the robots. This approach is an intuitive solution to the problem, yet it presents some serious drawbacks that reduce overall resilience and performance. The leader must have a representation of all the agents in formation and calculate destinations for all agents during a formation maneuver. This represents a large amount of computation as the number of agents increases and introduces a possibly dangerous amount of lag time between sensor input and reaction. The top down approach is also subject to a single point of failure; that point being the lead agent or central controller. If the leader or controller of the formation is disabled or malfunctions in some way, the entire formation can be lost due to lack of communication or faulty orders issued to the

formation.

The second type of solution is the bottom up approach, where decision-making is devolved from the central controller to the individual agents. Treating the formation as a collection of individual agents has several advantages. First, the computation of formation maneuvering is naturally distributed across the entire formation. For example, Mead's solution [19] presents the individual agents as cells in an n-dimensional cellular automaton.

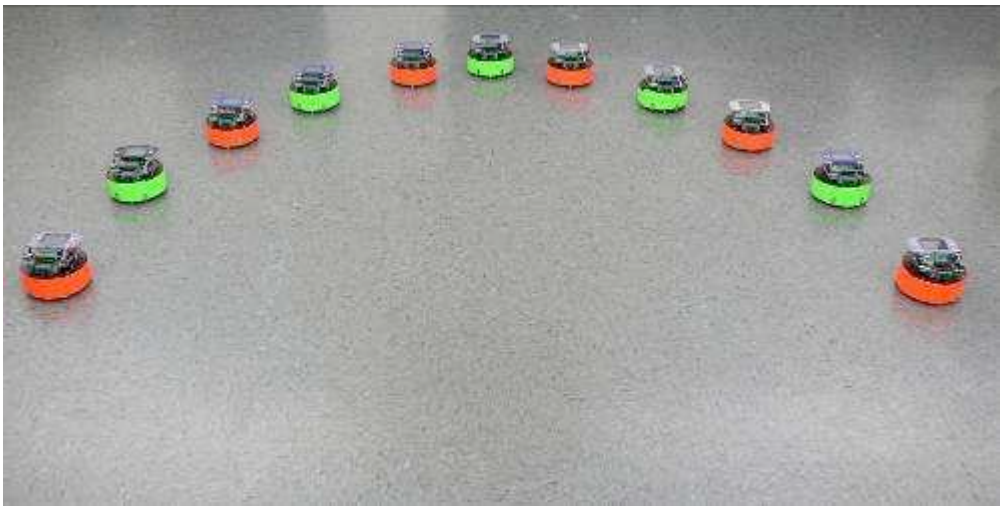


Fig. 2: A Parabolic Single Function Formation [20]

Each agent need only be aware of its location and orientation plus the state, location, and orientation of its cellular neighbors in order to calculate its new location and orientation, rather than requiring one agent to be aware of the location and state of each agent in the formation. A second advantage over top down solutions is the lack of a single point of failure. Since the formation has no leader, it

is not subject to loss of function resulting from the loss of the controller or leader.

While there have been a number of methods for maintaining robots in formation, there has been very little work done in coordinating the transformation of a *swarm* of robots into a formation. Methods such as Mead [19] and Fredslund and Mataric [10] assume that in a collection of robots the organization of robots into neighborhoods is already known, so negotiating a positional relationship between a robot and its neighbors can be done immediately. However, this is not practical for most applications. Consider for example, the Space Shuttle releasing a *swarm* of robots in orbit that must form an array for collecting solar power.

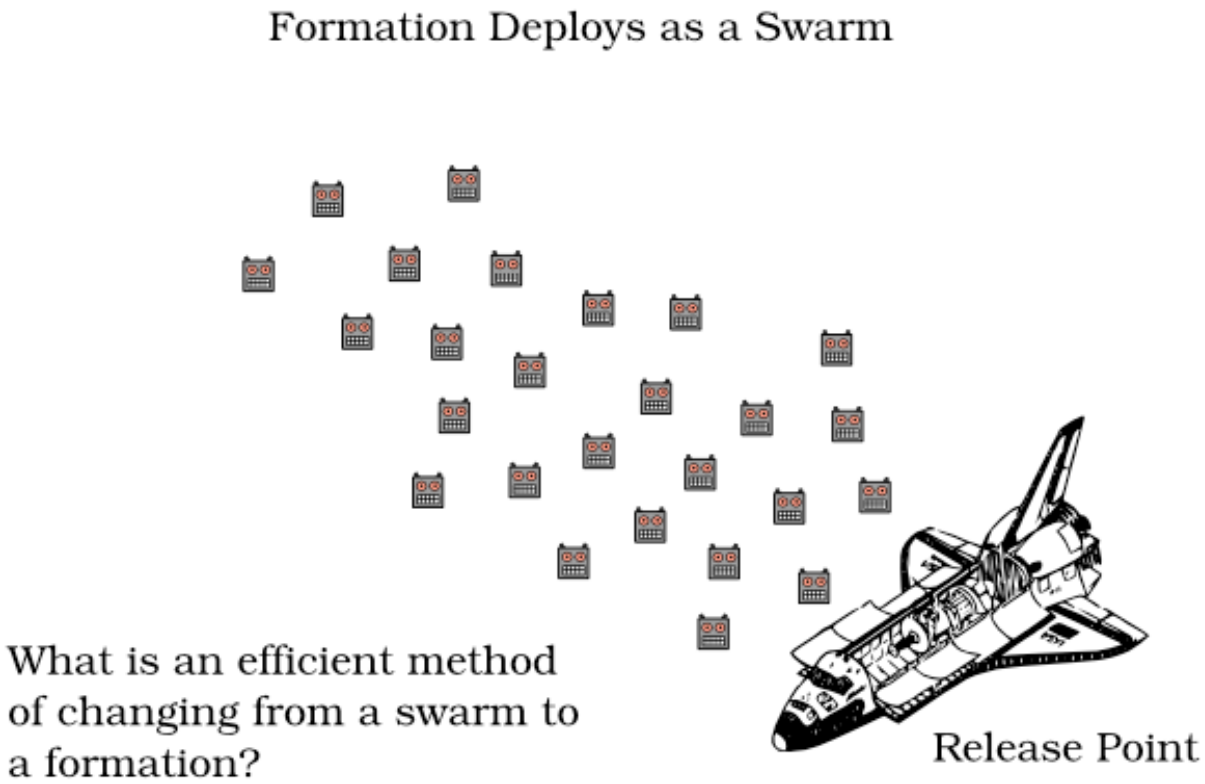


Fig 3: Deployment of a Formation from a Delivery Vehicle

The robots must first determine which among the *swarm* are their neighbors before they can negotiate a relationship. Developing an algorithm to dynamically form neighborhoods is a necessary step to allow a *swarm* to become a formation.

Phase Transition Metaphor

The focus of this work has been the development of a method for efficiently transitioning from a swarm to a formation. This transition will be referred to as a *phase transition* [23] since it has many parallels to the phase transitions observed in matter. For example, it is useful to think of a group of mobile robots with no particular programming for interaction with each other, aside from collision avoidance, as similar to matter in a gaseous state. That is, the volume and shape the robots take on, when taken as a collection, is determined by their surroundings. Robots traveling in a swarm are similar to matter in the liquid phase, since they have a fixed volume, but no fixed shape. A swarm will assume the shape of the boundaries of its surroundings but will have a fixed overall volume. Robots traveling in formation are similar to matter in the solid phase. Robots in formation resemble crystalline formations, with fixed volume and fixed shape. Robot formations may also have a repeating internal structure, which is situated in a repeating lattice.

In this work methods for initialization of a formation are explored using distributed auction algorithms. Auctions are typically used to solve resource

allocation problems, as in the "Contract Net Protocol" described by Smith [22] and in the work of Gerkey and Mataric [11]. Auctions provide a method of allocating resources, held by sellers, to agents that require those resources, the buyers. In the terminology of auctions, the seller makes use of an auctioneer, possibly the seller itself, and the buyer places bids on the items, which the auctioneer presents, to the sellers. An open call auction is one in which all of the bids proffered are known to all other bidders. A distributed auction is conducted by some group of sellers and buyers, usually assuming that all participants are trustworthy, in which all bids are issued and the auction settled without the aid of a central organizer. In the work of Gradwell and Padget [12], distributed auctions optimized for buyers are found to "increase the total number of items sold, cause a greater number of bidder's requirements to be met, and resulting run times are more consistent." A distributed auction algorithm may also be applicable to other formation problems, such as formation repair due to loss or disabling of some subset of the robots or re-forming a formation after obstacle avoidance or possibly merging two previously independent robot formations.

In Mead's [19] solution for control of large formations of robots, a "seed" is required to receive instructions and act as a catalyst for initiating orders. The seed does not directly relay orders to the rest of the formation, but rather, adjusts itself to the desired position and orientation, which causes its neighboring robots to adjust

their position and orientation to match, thus propagating the change across the entire formation. As part of the initial formation, a robot needs to be identified as the seed of the formation. One approach is to have a "Seed Election" that would enable a formation to select a new seed agent autonomously. A second approach is to allow the operator of the agents to select a seed at will. Any method for selecting a seed should consider minimizing the amount of maneuvering resulting from future position or orientation changes. This method must also seek to minimize the number of vertexes, when considering robots as nodes and a link to a neighbor as a vertex, in order to ensure the quickest propagation time for changes in position or orientation.

Purpose of the Project

The purpose of this project is to explore distributed autonomous algorithms for transforming a swarm of robots into an organized formation of robots. The project will build on top of Mead's [19] formation algorithm, CATALST, to create a distributed, auction-based method for establishing neighborhood relationships between homogeneous robots that define the formation. Parameters that will be used to evaluate the methods are total time to converge, average time to converge, total distance traveled, average distance traveled, total messages send, average messages sent, messages sent per time step, and generality of achieving different formations.

CHAPTER II

CONTEXT, BACKGROUND, AND LITERATURE REVIEW

Previous Work on Swarms and Formations

In the study of multiagent systems there are many approaches to regulating the interactions of agents; specifically, physically instantiated agents like mobile robots. Dudek, Jenkin, and Milios in [8] have proposed a useful taxonomy of Multiagent Systems of mobile robots. A *swarm* is defined as a "number of smaller, simpler robots", this in comparison with a single more complex robot meant to achieve a goal such as interplanetary exploration, [8]. Dudek et al [8] define a group of seven axes along which may be located any *swarm* or collective of mobile robots, depending upon the capabilities and makeup of the swarm [Fig. 4].

Axis	Description
Collective Size	The number of autonomous agents in the collective.
Communication Range	The maximum distance between two elements of the collective such that communication is still possible.
Communication Topology	Of the robots within the communication range, those which can be communicated with.
Communication Bandwidth	How much information elements of the collective can transmit to each other.
Collective reconfigurability	The rate at which the organization of the collective can be modified.
Processing Ability	The computational model utilized by individual elements of the collective.
Collective Composition	Are the elements of the collective homogeneous or heterogeneous.

Fig. 4: Seven Taxonomic Axes [8]

Swarms

These axes provide a framework within which the features of a group of mobile robots may be described. The first axis, Collective Size, is one area in which the techniques explored in this paper and those of Mead [19] differ from much of the existing work. In Fredslund and Mataric's [10] approach to formations, the largest formation attempted with physical robots was 4, with the largest simulated formation attempted being 10 robots. It is unclear from the paper [10] whether the method would scale well to larger numbers of robots. In Atay and Bayazit's [1] description of an "Emergent Task Allocation" method, no physical implementation was attempted, but the largest number of robots attempted in simulation was 30. Their method has a complexity that is limited by k, the number of neighboring robots to exchange information with, and the number of targets being tracked. The maximum number of robots supported by this method greatly depends upon the mission parameters. While their emergent task allocation greatly out-performs the centralized global optimization approach, it still took as long as 16 seconds to arrive at a solution for a single time-step in a simulation involving 20 robots tracking 10 targets [1]. In comparison to these implementations, the Space Solar Power project will require a formation of robots on the order of 10^3 robots [5]. This increase of two orders of magnitude will overwhelm these approaches by requiring too much time to calculate solutions.

Swarms vs Formations

It is important at this point to differentiate between a swarm and a formation. Mead [19] describes a swarm as "a massive collection that moves with no group organization". A formation, on the other hand, is defined as similar to a swarm, but maintaining some global structure [19]. This global structure can be defined as an extra frame of reference in which the member robots are situated. Formations have an extra frame of reference, the formation-relative position, which refers to the relative position of a robot within the formation. Swarm robots are limited to a global frame and individual robot frame.

To make a clear distinction between swarms and formations, a natural example of a swarm is a school of fish. The individual members of the school have no fixed relationship to any of the other members. The school undulates and changes size and shape as it maneuvers, with individuals changing position relative to one another. In Reynolds' [22] seminal work on swarms, he demonstrates that a simple set of rules governing and individual's positional relationship with its neighbors can attain overall swarm behavior [Fig.5].

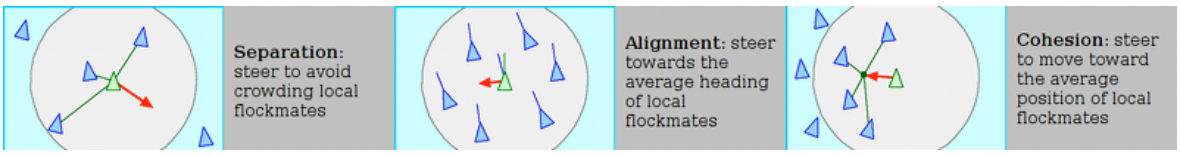


Fig. 5: Swarm Steering Behaviors [22].

However, it is useful to draw a distinction between swarms and formations. Where a natural example of swarm may be found in a school of fish, an example of a formation in nature is found in a "V" shaped flock of migratory birds. Such natural formations differentiate themselves from swarms in that any two individuals will maintain a relatively constant relationship to one another, with respect to alignment and cohesion [22], as the formation moves and maneuvers. The overall shape of the formation remains fixed despite changes in direction.

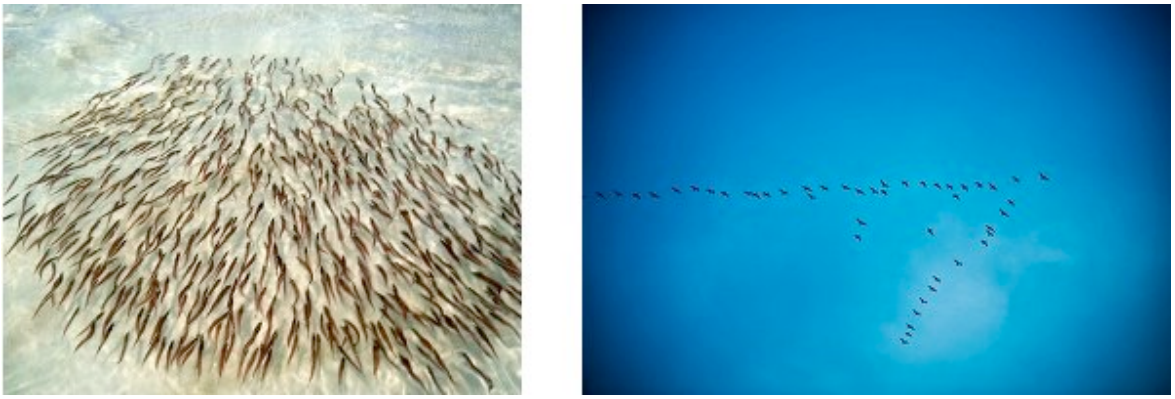


Fig. 6: Left – a Swarm, School of Fish; Right – a Formation, Flock of Geese

Formations

Work on mobile robot formations has focused on methods for maintaining formation relative positioning [10, 16, 17, 19, 20, 23, 25, 27]. Mead [19] has shown that a Cellular Automaton may be used to regulate an individual robot's behavior with respect to its neighbors, producing order as an emergent property among the group. Individual behaviors are determined by the desired outcome, that is, which formation is communicated to the seed robot. The seed orients itself to the proper

global position and heading as specified by the human operators, then those robots in the seed's immediate neighborhood orient themselves based on the current formation, to the seed. A key innovation of Mead [19] is the use of a "Robot Space" automaton in place of a "World Space" automaton.

The CATALST Approach to Formations

The World Space automaton presumes that the space in which the robots exist is gridded off into cells. These are the cells of the automaton, and the states of these cells are binary, either containing a robot or not. As Mead [19] points out, this model leads to some problems when applied to physical robots [Fig. 7].

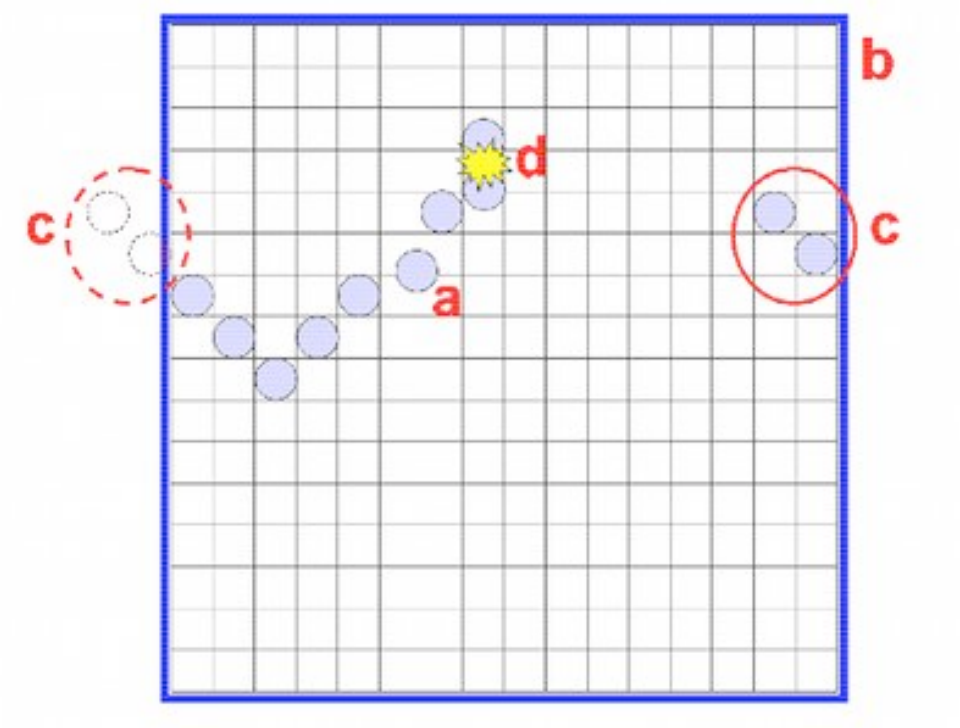


Fig. 7: Illustration of the Problems Encountered Using a World-Space Cellular Automaton: (a) Robot Not Located Inside Any Single Cell; (b) Automaton is Bounded; (c) Automaton Boundaries are Wrapped, Which Doesn't Reflect Reality; (d) Inter-Robot Collisions Occur [19]

An assumption implicit in the world space model of cellular automata robot control is that a robot must occupy one and only one cell at a time. This assumption is likely to be tested immediately upon deployment. Some amount of error is unavoidable, and will lead to robots overlapping cell boundaries over time, presenting a hazard to other robots and distorting the formation. This assumption, coupled with the fact that the cells must cover a finite area from which the robots cannot leave, means that the world space model is quite fragile. The biggest issue with the world space view of applying cellular automata to controlling formations of mobile robots is the implicit requirement that there be a single system, with knowledge of all local conditions and all locations of the robots, which manages the automaton. The member robots could themselves carry out the necessary processing to determine the state of the cell they currently occupy, but in a system where space itself is divided into cells, processing must also be done for the empty cells. This rules out a non-hierarchical approach.

The "Robot Space" or "Application Space" solution proposed by Mead, et al [19], models the robots themselves as the cells and the formation definition determines the distances and angles between the members of the formation [Fig. 8].

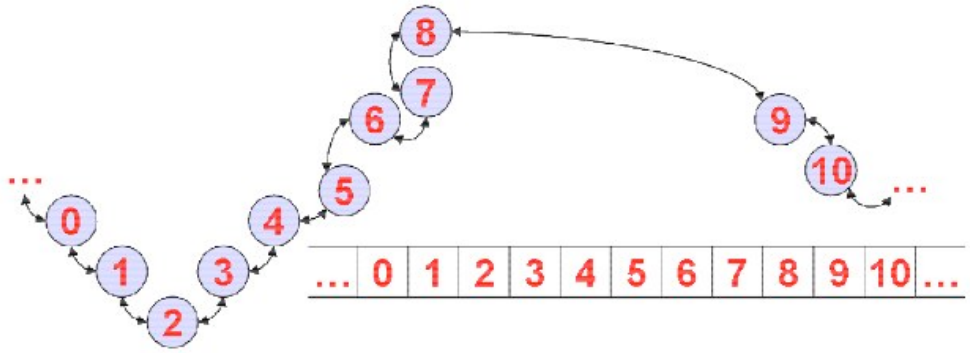


Fig. 8: A One-Dimensional Robot-Space Automaton [19]

Thus, this solution addresses the problem of requiring a single "all seeing" system to process the state of each cell and issue orders to the robots. The robot-space solution effectively distributes the management of the cellular automaton to each individual in the formation. This eliminates a large amount of communication as well. The robot-space solution also eliminates the problem of a limited grid size and is more tolerant of robot performance in the physical world in that it does not fall apart if the robot does not happen to be within one discrete gridded off portion of the world as known to a centralized controller. In terms of [8]'s axes, [19]'s solution would be a finite state automaton, along the Processing Ability axis.

The inter-agent communication required by the robot-space automaton is limited to a few key parameters. These parameters are summed up in the definition of a state, s_i. Mead et al. [19] define state formally as:

$$s_i = \{\ p,\ r_{des},\ r_{act},\ \Theta,\ \Gamma,\ t\} \qquad \text{Eq.1 [19]}$$

The state s_i is that of the cell c_i, in a one-dimensional automaton. The variable p stands for the position of the cell within the formation, or the formation-relative position, r_{des} represents the desired relations with neighboring cells, r_{act} represents the actual relationships with all neighboring cells, Θ represents the rotational error, Γ represents the translational error, with t representing the time step. The neighborhood of a cell c_i in a one-dimensional automaton can be defined as the set of the states $\{s_{i-1},\ s_i,\ s_{i+1}\}$. This requires a robot to transmit this information at most twice, once to each neighbor.

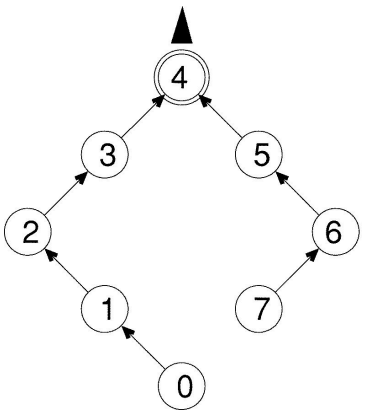
Other Approaches to Formations

Sawada et al. [23] developed a method to control conference room tables with the goal of organizing them for a dynamic schedule of various events. This method utilizes an overhead camera and infrared LED's in order to sense the location and orientation of the table robots. The camera is linked to a central control agent that calculates the paths and final locations for the table robots.

The Distributed Flight Array of Oung et al. [21] represents a distributed approach to formation control and initialization. The solution utilizes a large light that shines down on all the robots that will be included in the formation. The individuals move in order to maximize the reading on their photometers. This causes the individual agents to gather at the center of the cone of light projected by the light source, where the individuals physically dock with one another, making

connections that are held fast. The formation makes further calculations to determine the balance of the torque generated by the main propellers of the individual agents.

The approach of Mead et al. [19] has some advantages over previous formation control schemes. For example, Fredslund and Mataric's [10] use a “friend sensor” in order to track one robot in formation to which the individual has a defined relationship. This sensor, which was implemented as a combination of a laser range finder and a color camera, is used to recognize the presence and orientation of a neighbor, which is determined prior to deployment. Distance and angles between neighboring robots are determined by the formation definition. The angle between individuals is determined by the formation definition and maintained by panning the friend sensor so that the resulting angle between the individual and its friend is also adjusted by the same angle. Because the follower robots must always reference a leader, a drawback of using the physically panning sensor with a rotational range of 180 degrees in the Fredslund and Mataric [10] solution, is that “frontally concave” formations, such as a “J” or “U” shape are not possible. This limitation could rule out this solution for use in Space Solar Power [5] applications that may benefit from the use of a parabolic formation. A second drawback of the Fredslund and Mataric [9] solution is the requirement that all formation arrangements be either one dimensional or folded variants of one-dimensional formations. [10; Fig. 9].




```
n = ⌈N/4⌉
if lessThanMe < ⌊N/2⌋ − n
  θ = 45
else if lessThanMe < ⌊N/2⌋
  θ = −45
else if lessThanMe > ⌊N/2⌋ + n
  θ = −45
else if lessThanMe > ⌊N/2⌋
  θ = 45
```

Fig. 9: An Example of a Folded 1-Dimensional Formation and Required Function to Calculate the Friend Angle. The Numbers at the Nodes Represent the Id's of the Members of the Formation, with 4 being the Conductor. Fredslund and Mataric [9]

This does not preclude the formation of rectangular or hexagonal grids, but does imply that computational requirements might be higher for individual robots and that error in friend angle calculations or alignment will be magnified and effect the entire formation if the error were near the "conductor" of the formation.

A multi-function approach to formation definitions has been proposed by Mead et al. [19, fig. 10].

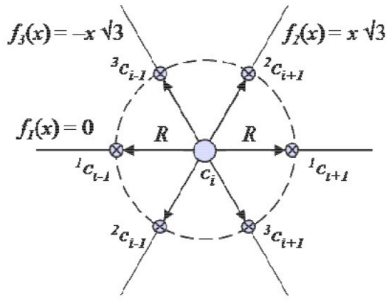


Fig. 10: Multi-Function Formation with Three Formation Definitions [19]

The proposed method is identical to the one-dimensional approach, but with an expansion of the formation definition F from a single function to a series of M mathematical functions to describe the relations of neighbors in the formation. However, if the functions [Fig.10: $f_{1,}$ $f_{2,}$ and, f_3] are used to calculate the desired relationships for this automaton, without taking further steps to adjust for two-dimensionality, an odd "six armed" shape is created which is not what is expected [Fig.11 B].

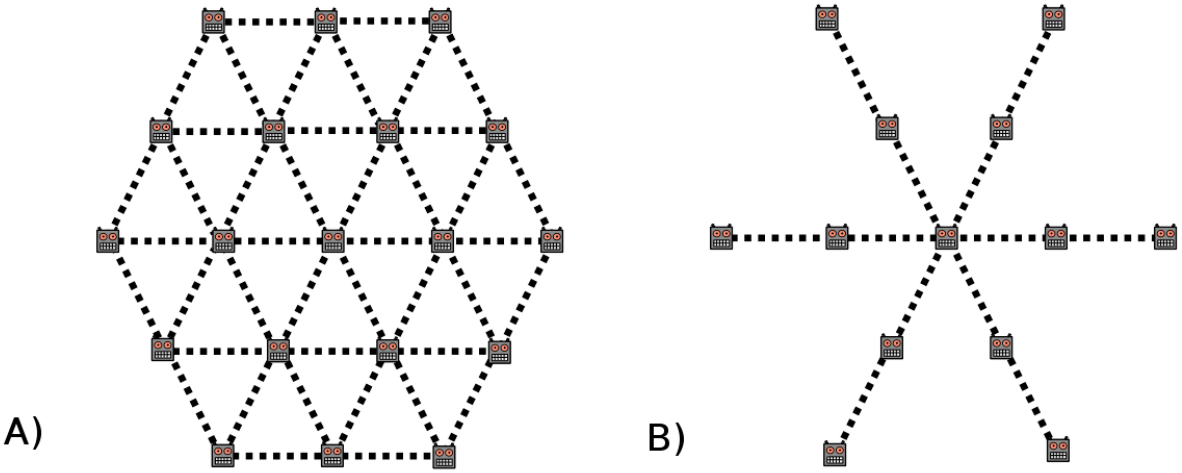


Fig.11: A) Desired Shape of a Hexagonal Lattice Formation, B) Actual Shape Resulting From 3 Formation Definitions, Implemented as One-Dimensional CA's [19]

A hexagonal lattice is the expected result [Fig.11 A], and this can be achieved if the formation relative position used to calculate the desired relationship to a neighbor is altered so that each individual sees itself of the center of the formation and has a neighborhood of six robots.

Previous Work on Distributed Auctions

A method for coordination among groups of mobile robots that has received much attention is that of the auction. Implementations of auctions for mobile robot coordination take many forms, from the high-level framework for communicating the terms of auctions, the "contract net protocol" [24] to the more specific application of auction algorithms to the allocation of tasks to a heterogeneous group of mobile robots [11].

The very nature of auctions suggests the development of distributed auction algorithms. The most basic form of parallelism in auction algorithms is that of the simultaneous calculation of bids by buyers in the auction. In the work of Smith [24], the benefits of a distributed solution are elaborated upon at length. These benefits mirror those of decentralized control in many ways, including the lack of a single point of failure, the spreading around of information so that more information is available to more individual nodes, and the ability to route around congestion.

Problems With Distributed Auctions

Lagoudakis et al [15] have pointed out that many task/target allocation cost minimization problems in mobile robotics are NP-Hard. This implies that the auction solution, especially the Sequential Single Item (SSI) auction, which is "a series of auctions in which a single item is auctioned off" with as many auctions as

there items to be auctioned, is not guaranteed to find an optimal solution [14].

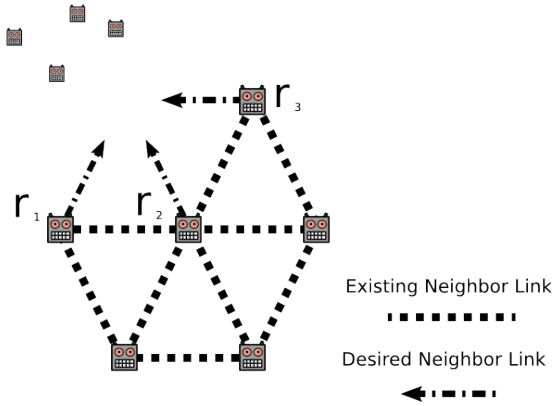


Fig 12: Three Auctions Reference the Same Position.

An issue that must be dealt with when considering the use of auctions to initialize a formation, is the possibility that two or more auctions will exist for the same position in the formation, but originating from two or more robots [Fig. 12].

An approach to solving the multiple auctions of the same item, in the case of this project, that single item being a single formation relative position, is that explored by Sujit and Beard [26], where the authors are solving the problem of multiple UAV's spotting the same target, and auctioning off the attack of the same target, multiple times by different UAV's. The method explored in [26] was that of staging a two round auction process. In the first round, any agents that have encountered a target within their sensor range are to broadcast a validation request to all agents inside communications range. If a UAV receives a validation request, it evaluates that request against any targets it is detecting and also against any other validation requests received from other agents. In this manner, even if two UAV's

are able to spot the same target, and happen to be unable to communicate with each other, but both can communicate with a third UAV, the overlap in target allocation can be detected [26].

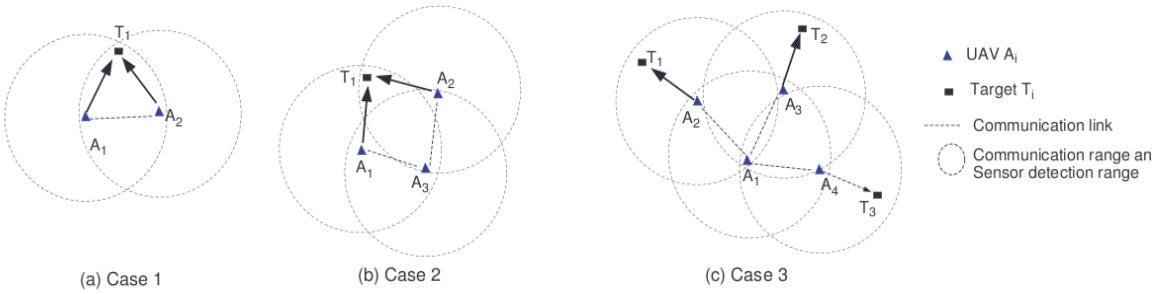


Fig 13: Three Cases in Which Ambiguities Can Arise [26]

In figure 13, three cases are presented by [26] in which there is ambiguity as to which UAV should auction a target, whether or not the targets can be validated, and how to make decisions regarding which targets to bid on. In case 1 [fig 12a], UAV's A_1 and A_2 can both sense target T_1. The validation round in the auction process resolves this ambiguity by allowing A_1 and A_2 to report each other's target as non-unique. The algorithm described by [26] gives ownership of the auctioning responsibilities to the UAV with the shortest Dubins path to the target, Dubins path being the shortest possible path on a plane between two points, along which a nonholonomic vehicle with a known minimum turning radius can travel. This case is similar to the situation portrayed in figure 12 between R_1 and R_2. Disambiguation can be accomplished in the same manner used by [26]. In case 2 [fig 12b], A_1 and A_2 can detect target T_1, but cannot communicate with each other, they

can however, communicate with a third UAV, A_3. This situation is resolved in the validation phase within A_3, which will compare the validation requests from A_1 and A_2 and find that both requests are the same.

In a square lattice formation of robots overlapping auctions can occur. Additionally, robots that are not neighbors of each other will hold these auctions. Robots not located in the same neighborhood cannot be assumed to be able to communicate with one another outside of the broadcast framework of auction announcement and bid communication. A solution to disambiguate the proper auctioneer of this position must be found in order to generalize the use of auctions across single and multi-dimensional formation definitions.

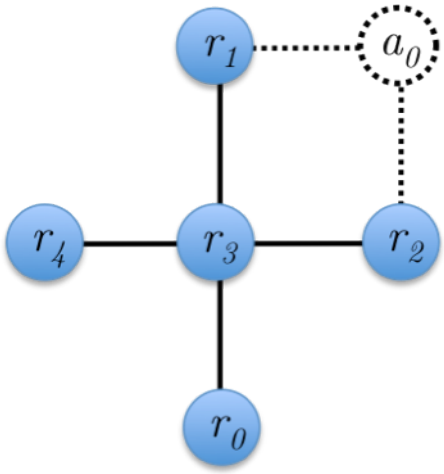


Fig.14: Square Lattice Overlapping Auction Problem.

This situation is similar to case 2, in that a robot may act as a go between to settle the dual listing of the relationship in auctions. In case 3 [fig 13c],

there are three UAV's, A_2, A_3, and A_4, which can each detect a single target, T_1, T_2, and T_3 respectively.

Other Approaches to Distributed Auction-Based Initialization of Robot Formations

At the time of writing, the only published method for autonomously initializing a formation of mobile robots, or causing a swarm of robots to undergo a phase transition into a formation, known to the authors, is that demonstrated by Lemay et al. [16]. The method is built around a formation control architecture, which is very similar to that of Fredslund and Mataric [10], in that it is implemented with a follow-the-leader method, with each robot using a sensor to track the lead robot.

The autonomous initialization method of Lemay et al.[16] begins with each robot executing a 360° turn in order to find via camera, which robots are within its sensing range. The robot then places a row containing a column with the ID of each visible neighbor on a "visibility table". Once this phase is complete, the robots exchange their visibility information, and new rows are added to each robot's visibility table, representing the robots visible to each of the other robots. Each individual then uses this data, assumes it is the conductor, or leader, of the formation, and does a "bounded depth-first search" looking for the most efficient configuration [16]. Since each individual performs this search, the algorithm implements a distributed depth-first search. When the result of the search is

obtained, the individuals each broadcast their best configuration, and the lowest cost configuration is chosen and implemented [16].

While the method employed by Lemay et al [16] does utilized distributed auctions in order to dynamically assign neighbors in a formation of mobile robots, the approach differs substantially from that employed in this work. While the work of finding the best starting configuration is distributed among the individual agents, it is still calculated in a top down method, with each agent being assigned a subset of state-space of the problem to work on. So the choice of distributing the work of calculation among the agents of the swarm is purely conventional. The solution sought by this algorithm is not emergent. The problem with the [16] approach remaining a top down method is that it requires each agent in the collection to have knowledge of all other agents. As the number of agents increases, the demands on computing power and space will increase greatly. Also, as the agents are distributed further away, unified communication becomes difficult. Even with message passing, the complexity of this solution is great.

The solution developed for this project, which will be described in the next chapter, is an emergent solution. While this represents an approximation of the ideal solution, it does distribute the problem in such a way that a central control agent cannot mimic it. The solution relies upon the interactions of multiple agents, simulated or actual, in order for the desired outcome to emerge.

CHAPTER III

DISTRIBUTED AUCTION-BASED INITIALIZATION OF FORMATIONS OF MOBILE ROBOTS

In this chapter, two distributed auction-based methods for initializing a formation of mobile robots are discussed. The first, the *push auction method*, casts the cells of a formation as auctioneers and the unassigned robots as bidders in which winning bidders are added to the endpoints of the formation. The second method is the *insertion auction method*, which reverses the roles and casts the unassigned robots as auctioneers and the cells of the formation as bidders in which the winning bidder moves along the formation function and inserts the auctioneer into the position just vacated by the bidder.

Assumptions

With both approaches there are several underlying assumptions that are vital to understanding the environment and conditions in which the solutions proposed are made.

It is assumed that the collection of robots to be organized into a formation is homogeneous. Each robot is assumed to be able to carry out the duties expected of it at any position in the formation to an equal capacity.

It is assumed that the robots to be organized are holonomic robots. That is, it is assumed that each robot has the capability to move in any direction without

first translation through the *x, y,* or *z* axes. Rotation without translation is allowed. This assumption has been made in order to simplify the implementation and analysis of the auction algorithms, however, the auction methods discussed here are generalizable to nonholonomic robots as well.

It is assumed that each robot is capable of communicating with all other robots within a defined radius via wireless means. For the purpose of implementing an effective simulator, it is also assumed that communications are transmitted and received without error, as such issues as error correction and detection are beyond the scope of this work.

It is assumed that each robot possesses a reactive control architecture and the appropriate sensor suite in order to permit it to avoid obstacles independent of the formation architecture or the initialization method.

It is assumed that all sensors work instantaneously and without error. Again, this is done in order to facilitate the implementation of the auction algorithm rather than reproducing work in the area of sensor error estimation and correction, which is beyond the scope of this work.

While the method explored in this work are applicable to robots operating in a three dimensions, perhaps in an underwater setting or the vacuum of space, the simulator is currently restricted to two dimensions. The topology of the two-dimensional space in which the simulation occurs is an infinite flat plain. Obstacles

introduce an interesting dynamic into both local path planning for individual robots and for formation initialization.

In addition to the previous assumptions about the robots, their capabilities, and the environment in which they are located, there are some assumptions regarding the status of the robots with respect to the CATALST [19] architecture.

It is assumed that the formation definition contains one function. While the methods discussed below have been considered for use in multi-function formation definitions, it is assumed that the formation is a single function formation.

It is also assumed that before either of the distributed auction methods are applied that a formation of one or more cells has already been established. This assumption is made in order to distinguish this work from the work on seed selection algorithms being done by Beer et al [4]. The individual agent chosen to be the seed affects the overall efficiency of the process of initializing a formation of mobile robots significantly, but such concerns are beyond the scope of this work.

Push Auction Method

The *push auction method* casts *cells* as auctioneers selling their neighboring formation-relative positions. Robots that are not yet part of the formation are cast as bidders, and newly added robots become cells that are "pushed" onto the ends of the formation.

Push Auction Algorithm

```
    /* Process Running on Cell */
    /* Determine if an auction should be held, if so announce it */
1   if n < n_max and Γ ≈ 0 then
2       broadcast(A(p_j))
3       t_a := COUNTDOWN_TIME
        /* Accept bids and decrement auction timer */
4       while t_a > 0 do
5           if front(incoming_messages) = B(p_j) then
6               enqueue(bid_queue,B(p_j))
7           t_a := t_a – 1
8       End
9   End
    /* If no bids received, restart */
10  if size(bid_queue)=0 then exit
    /* Sort received bids, select lowest, announce winner */
11  sort_ascending(bid_queue)
12  b_w := front(bid_queue)
13  broadcast(p_j, state(b_w->ID))
    /* At this point, winning bidder becomes a cell */

    /* Processes Running on Unassigned Robot */
    /* Auction Watcher */
1   while front(msg_queue) ≠ A() do
2       get_messages(msg_queue)
3   end
4   foreach A(p_j) ∈ msg_queue do
5       broadcast(B(p_j))
6   End
    /* Transformation Watcher */
7   while front(msgqueue) ≠ {p_j, state(self->ID)} do
8       get_messages(msg_queue)
9   End
10  transform_robot_to_cell(p_j s_j)
11  kill auction_watcher process
12  kill transformation_watcher process
```

Fig. 15: Push Auction Algorithm

The push auction algorithm is described in [Fig. 15] above. This algorithm is divided up into two key portions: the process to be run by a cell and the process

to be run by an unassigned robot. It is essential to the distributed nature of this algorithm that both processes run concurrently. Communication between the processes is accomplished by means of the publish/subscribe method discussed in Chapter IV: Implementation.

It is assumed that a seed cell has already been selected, and that the formation definition has been transmitted to this cell. This example uses the function of f(x) = 0 as the formation goal, which is a horizontal line, with c_{seed} as the seed cell [Fig. 16].

Fig. 16: Example of a Formation With a Function of f(x) = *0*.

Initially the robots are in a swarm, a collection of four robots (r_0, r_1, r_2, r_3) scattered in a random distribution, with a robot, c_{seed} designated as the seed [Fig. 17]. The push auction method begins when the seed receives a signal from a controller outside the formation to begin. The seed takes this signal to be a formation change request that establishes the parameters for the formation. Once this is received, the seed calculates the desired positions of its neighbors. The sole member in the formation begins the auction process if the requirements for auctioning are met [Fig. 15, line 1].

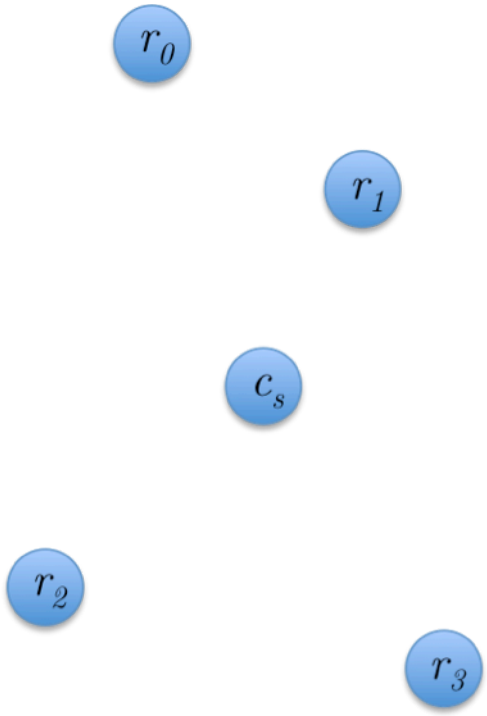


Fig. 17: Four Robots Around a Formation Containing a Single Cell c_{seed}

In the push auction method there are several prerequisites that must be met before the cell in question may initiate an auction. First, a cell must have an incomplete neighborhood ($n < n_{max}$)[Fig. 15, line 1]. In the example of a single function formation definition this means that a cell must have fewer than two neighbors. In the case of the seed, c_{seed} has zero neighbors as it is the sole member of the formation. In the example, c_{seed} has no neighbors, and can auction off the formation-relative positions to the left and the right. Second, the cell in question c_i must have a translational error Γ magnitude that is less than some arbitrary amount set by the formation controller. In order to minimize the distance traveled, disallow

auctioning until the condition $\Gamma \approx 0$ is true. While this rule applies to all cells including c_{seed} it does not limit the seed quite as much since the seed will only ever have a translational error above zero when the formation controller has issued new global coordinates for the formation to move to and c_{seed} has yet to arrive at that new location.

If there is no restriction on the Γ of a cell wishing to initiate an auction, there will be a chain reaction of auctions before the robots have time to move into their new positions in the formation. This has two drawbacks vis-à-vis the push auction algorithm. First, each auction held by a cell with a relatively high Γ will result in bids that are based upon the current position of the cell and not the desired position of the cell. This results in a higher total distance traveled [Fig. 18]. For example, r_i wins auction a_i. At this point, r_i becomes c_i and if there is no restriction on cells holding auctions until $\Gamma \approx 0$, then the auction a_j* will be held.

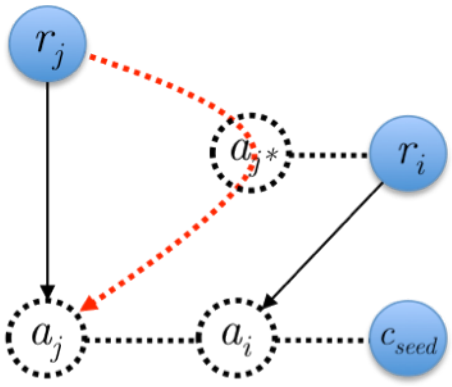


Fig. 18: No Restrictions on Γ Results in a Longer Path for r_i

The robot r_j will win this auction and as it moves to take its position it becomes c_i. This new cell will be moving towards the location of a_i. This will yield the much longer dotted, curved path to a_j. However, if auctions are restricted until cells have reached a near zero Γ, then the auction a_j will be held and c_j will move along the black colored path, which is a great deal shorter.

The second drawback of this chain reaction of auctions is that each robot that is not yet a member of the formation will be moving into position once it has been won an auction. This may not immediately seem like a problem, but it creates a condition where many robots in motion in the same area increases the likelihood of collisions and increases the total distance traveled by causing collision avoidance behaviors to be activated in the various individual robots.

At this point, if c_{seed} meets the above requirements, it will prepare an *auction announcement* [Fig. 15, line 2]. An auction announcement for a push auction of the formation-relative position p_j is expressed as $\mathrm{A}(p_j) = \{s_{seed}, p_j\}$ where s_{seed} is the state of the seed and p_j is the formation-relative position up for auction [Fig. 19]. Once the auctioneer transmits the auction announcement, a timer is set that counts down [Fig. 15, line 3]. This timer is the amount of time that the auction will remain open.

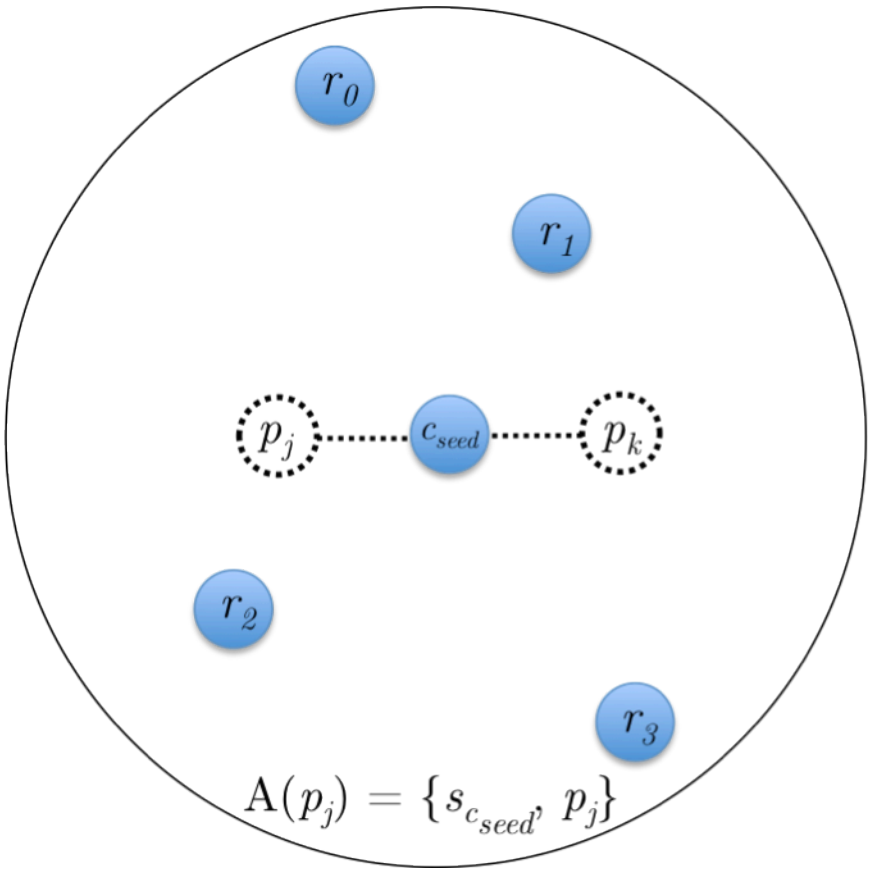


Fig. 19: The Unassigned Robots Within Communication Range of c_{seed} Receive the Auction Announcement.

When an unassigned robot receives an auction announcement it immediately tests to see whether or not it can detect the auctioneer with its sensors [Fig. 15, lines 1-6]. Detection in this context refers to the ability to discern the range and bearing. The identification of another robot can be accomplished by many methods. One of the methods explored by Mead et al. [19] include the use of a multi-colored "face" that uniquely identifies an individual agent. If the unassigned robot receiving the auction announcement can detect the auctioneer, it will then determine

the range and bearing of the auctioneer relative to itself. The problem of finding the range to another robot is one that has been explored by Heil [13] via the use of a "trilaterative localization system for small mobile robots in swarms." Therefore, per the previous assumptions, it is taken that the unassigned robot receiving the auction announcement is capable of detecting the range and bearing of the auctioneer.

Utilizing the sensor data describing the range and bearing of the auctioneer, the bidder may prepare a bid for the formation-relative position being auctioned p_j represented as $\mathrm{B}(p_j) = d$ [Fig. 15, lines 4-6]. The bid is the distance d from the bidder, as calculated by the path planning method employed by the individual robot preparing the bid, to the position being auctioned (p_j). In order to calculate the distance factor d, the bidder must utilize the state of the auctioneer s_i to calculate the distance to the position being auctioned. This is not readily obtainable by means of the sensor suite due to the fact that any detectable object would not currently occupy the position being auctioned. In order

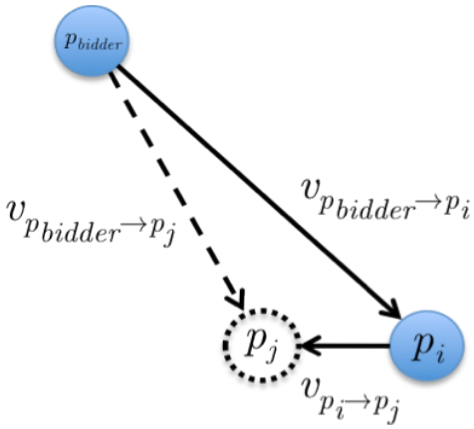


Fig. 20: Finding the Euclidean Vector Between the Bidder and the Formation-Relative Position Being Auctioned $v_{p_{bidder}\to p_j}$.

to calculate the Euclidean vector from the bidder to the formation-relative position being auctioned $v_{p_{bidder}\to p_j}$, which is unknown, the bidder must add the two known Euclidean vectors $v_{p_{bidder}\to p_i}$ and $v_{p_i\to p_j}$. This vector addition is illustrated in [Fig. 20] above.

The distance factor d will be determined by finding the magnitude of the vector between the bidder and the formation-relative position up for auction p_j represented by $||v_{p_{bidder}\to p_j}||$. The bid is then composed by the bidder and sent by a direct message to the auctioneer [Fig. 21].

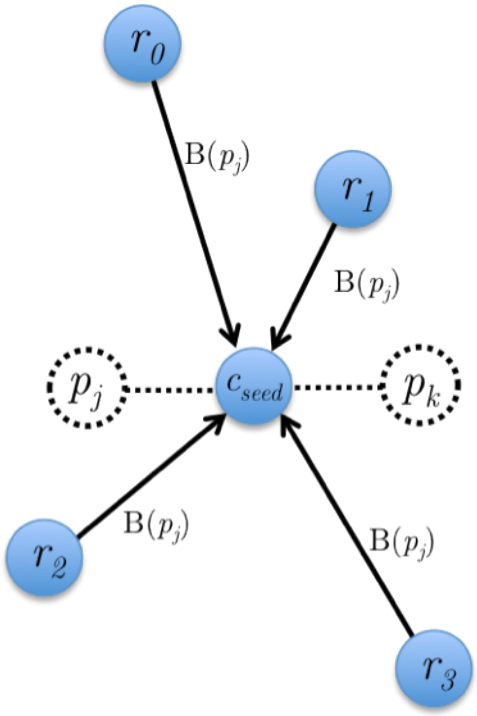


Fig. 21: Unassigned Robots Submit Bids to the Auctioneer.

Once the auction timer of the auctioneer has reached zero the auctioneer will no longer accept incoming bids [Fig. 15, lines 4-8]. Any bids received after this point will be dropped without notification to the sender. The auctioneer collects the received bids and search them for the minimum bid [Fig. 15, lines 11,12]. By selecting the minimum bid the auctioneer insures that the bidder closest to the formation-relative position, with some consideration for the weights in the bid function, will be chosen.

After the auctioneer has selected the appropriate bid, the winning bidder is notified [Fig. 15, line 13]. A new *cell* c_j will be generated [Fig. 15, lines 10-12] that

the winning bidder will now occupy [Fig. 22].

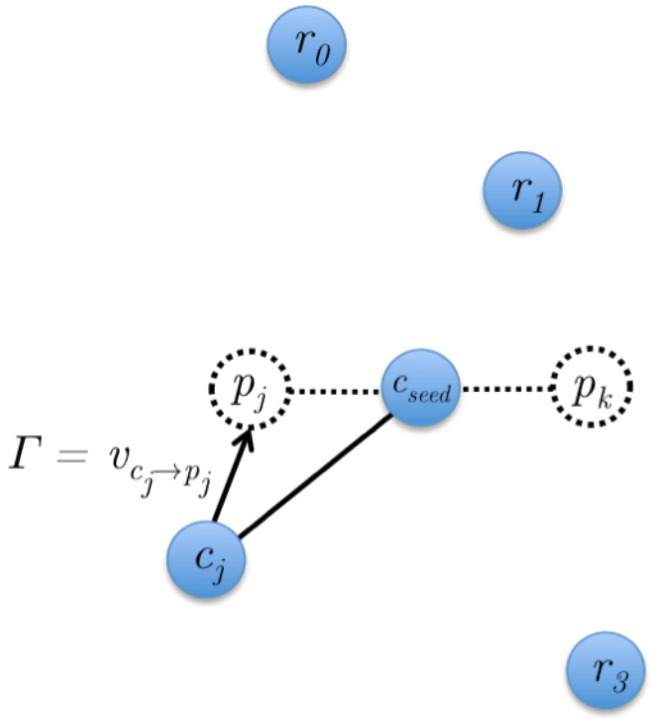


Fig. 22: c_j is created with an initial translational error equal to the original d of c_j.

The new cell c_j will now have a neighborhood containing c_i and will have a Γ exactly equal to $v_{p_{bidder} \to p_j}$ and will begin operating exactly as any other cell in the automaton [Fig. 22]. This implies that c_j will now begin moving in order to cause the condition $\Gamma \approx 0$ to be true. However, once c_j is created, the push auction cycle is completed.

Once c_j has achieved $\Gamma \approx 0$ and the other conditions for auctioning are met, c_j may hold an auction of its own in order to fill the empty formation-relative position to the left of it. See the end result of the push method auction in [Fig. 23] below.

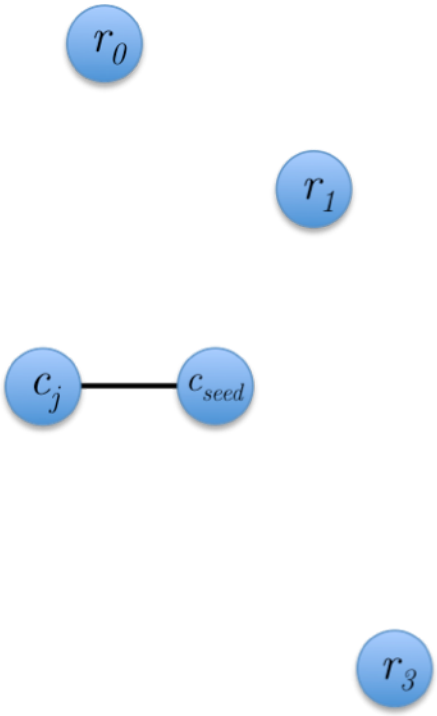


Fig. 23: The collection of robots after the completion of one push auction.

Insertion Auction Method

The insertion auction method reverses the roles of cells and unassigned robots in the push auction method and casts the unassigned robots as auctioneers of their services to the cells, which act as bidders. As unassigned robots conclude auctions, they become cells and are inserted into the formation, taking the formation-relative position of the winning bidder.

In order to explain the insertion auction algorithm an example is provided. Using the same example as before, the goal is a formation with a function of f(x) = *0*, which is a horizontal line, with c_{seed} as the seed cell [Fig. 24].

Fig. 24: Example of a Formation With a Function of f(x) = *0*.

This example also contains a collection of unassigned robots (r_0, r_1, r_2, r_3) scattered randomly about the location of c_{seed}.

Insertion Auction Algorithm

```
    /* Process Running on Unassigned Robot */
    /* Determine if an auction should be held, if so announce it */
1   broadcast(A(r_j))
2   t_a := COUNTDOWN_TIME
    /* Accept bids and decrement auction timer */
3   while ta > 0 do
4       if incoming_message(B(r_j)) then
5           enqueue(bid_queue,B(r_j))
6       t_a = t_a - 1
7   end
    /* If no bids received, restart */
8   if size(bid_queue)=0 then exit
    /* Sort received bids, select lowest, announce winner */
9   sort_ascending(bid_queue)
10  b_w := front(bid_queue)
11  broadcast(r_i, b_w->ID)
12  transform_robot_to_cell(self, T(c_i))
    /* At this point, r_i becomes a cell */

    /* Processes Running on Cell */
1   while front(msg_queue) ≠ A() do
2       get_messages(msg_queue)
3       foreach A()∈ msg_queue do
4           if distance_to(A()) < a_nearest then
5               r_i := A()->FROM_ID
6           end
7       end
8   end
9   broadcast(B(r_i))
```

Figure 25: The Insertion Auction Algorithm.

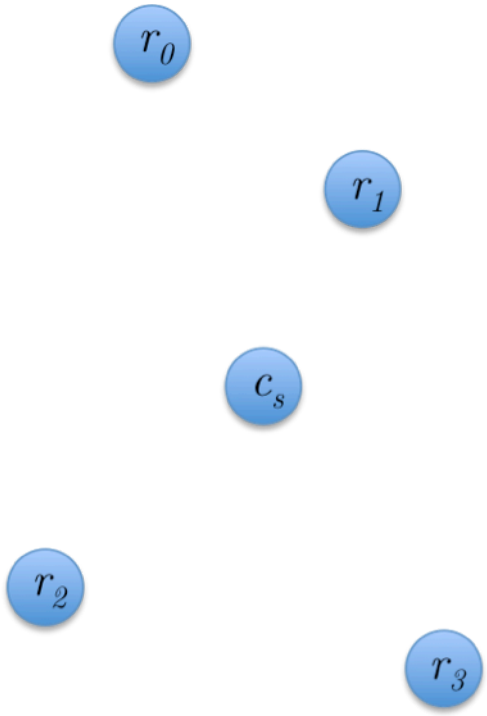


Fig. 26: An automaton containing one cell c_{seed} and a collection of unassigned robots begins the insertion auction example.

Insertion auctions may begin once there is at least one cell. This implies that the cell is the sole member of a formation with a formation definition. As previously discussed, this single cell must be the seed c_{seed}. Once the unassigned robots detect the presence of a cell, they will each begin holding auctions by transmitting auction announcements [Fig 25, line 1, Fig. 27]. Similar to the cells in the push auction, the unassigned robots have a countdown timer that determines the length of time that the auction will be open [Fig. 25, lines 3-7].

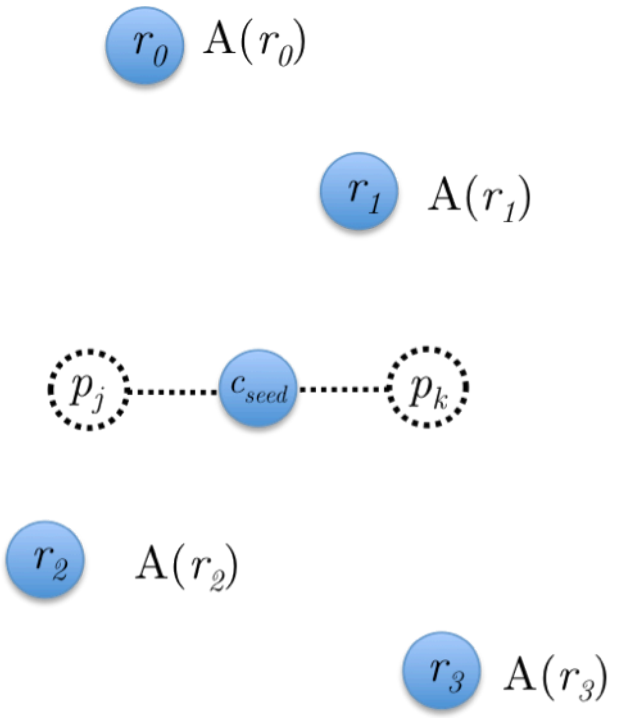


Fig. 27: The unassigned robots broadcast auction announcements.

When a cell receives an auction announcement it will check to see if it can detect the range and bearing of the auctioneer [Fig. 25, line 4]. The cell will then select the announcement originating from the nearest auctioneer and prepare a bid to return to this auctioneer. The bid function for a cell in an insertion auction is $\mathrm{B}(r_{auctioneer}) = d$ [Fig. 25, line 9, Fig. 28]. The cell will only bid on the nearest auction since it is desirable to minimize the number of messages sent and bidding on auctions further away than the nearest is an activity not likely to yield efficient behaviors.

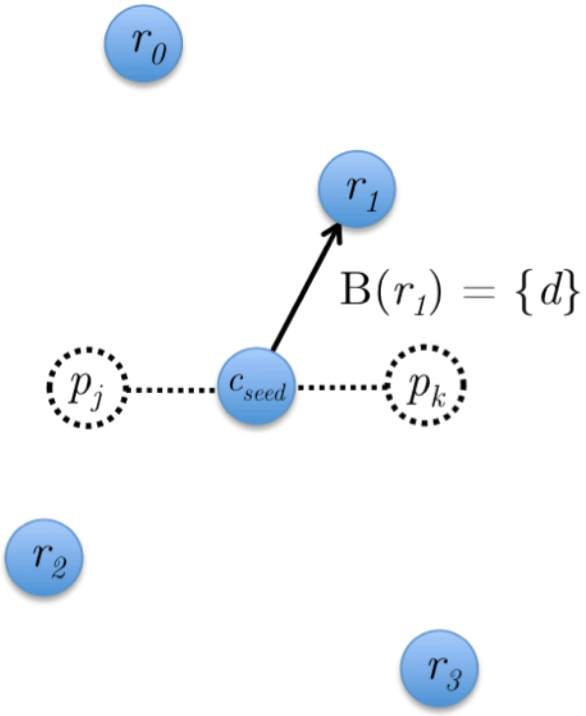


Fig. 28: The cell c_{seed} proffers a bid to r_1.

The auctioneer collects bids until the countdown timer has reached zero and the auction is closed [Fig. 25, lines 3-7]. If the auctioneer has received no bids, the auction is closed and the insertion auction cycle for that auctioneer is completed [Fig. 25, line 8]. If there were bids received, the auctioneer will now resolve the auction by selecting the lowest bid from the received bids [Fig. 25, lines 9-12].

At this point the resolution of the insertion auction diverges between the special case of c_{seed} winning the auction and any other cell winning the auction. If c_{seed} has won the auction, the auctioneer will move to the formation-relative position to the right or left of the seed, depending on whether or not one of those positions is

open. If both positions are taken, then c_{seed} will select a side to send the auctioneer to, based on the direction that has fewer cells [Fig. 29]. In the event that both directions have the same number of cells, c_{seed} will choose a direction at random.

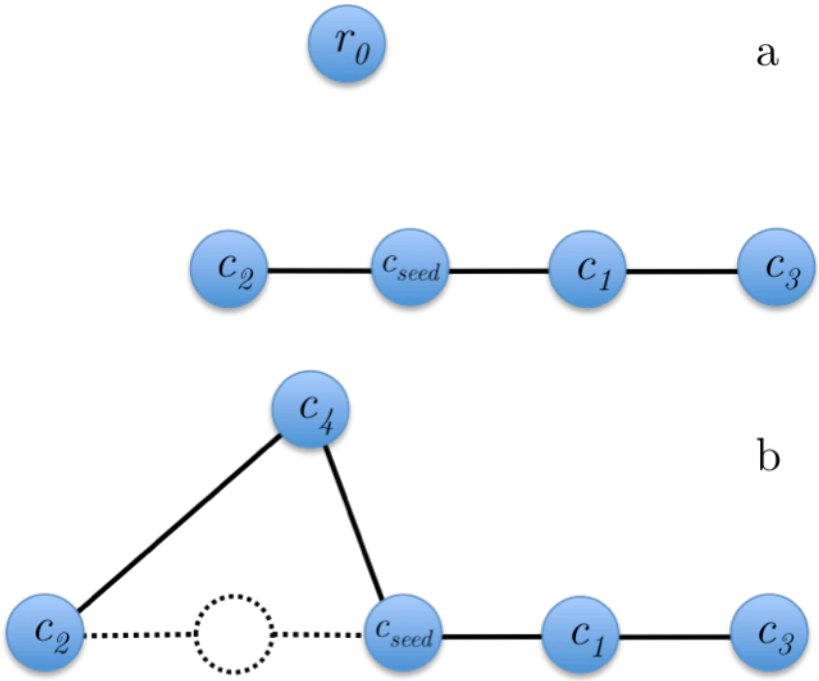


Fig. 29 a: c_{seed} wins an insertion auction from r_0 b: the new cell is inserted to the left of c_{seed}.

In [Fig. 29] the example of c_{seed} winning an insertion auction while also having a full neighborhood results in c_2 becoming a neighbor of c_4 and breaking its previous neighbor relation with c_{seed}. The new cell c_4 becomes a neighbor of c_2 and c_{seed}.

When a cell other than the seed wins an insertion auction, the winner will give the new cell its previous formation-relative position and will move one radius

away from the seed along the function on which the auction occurred. Any cell that is situated such that the new cell is between it and c_{seed} along the formation function, will add one radius to its formation-relative position, moving it further from c_{seed}. In the above example [Fig. 29], if c_2 had won the auction, the auction would have been resolved in an identical way.

CHAPTER IV

IMPLEMENTATION

Software

In order to evaluate the push and insertion auction methods, the simulator built by Mead [19] was extended to allow auction based formation initialization. Mead's simulator was built in c++, compiled for the Windows XP operating system, utilizing the OpenGL graphics library. Mead's code for this work was ported to compile in GCC on the GNU/Linux operating system with the X server windowing subsystem and OpenGL.

In order to allow for pipelining of trial runs for data collection, the simulator's existing facilities for parsing command line arguments was extended to cover the new parameters introduced by the addition of auctioning to the simulator. In addition to command line parsing, the simulator was extended in order to collect and dump pertinent data to files in the current working directory.

A series of Perl scripts was developed in order to manage the pipelining of the trial run process. The primary script contains a range of parameters that the script maps to runs of the simulator. At the completion of a run, the script create a new directory in the ./data directory, which is named based on the parameters used to run the simulator, and move all of the associated output files to that directory.

Screenshots

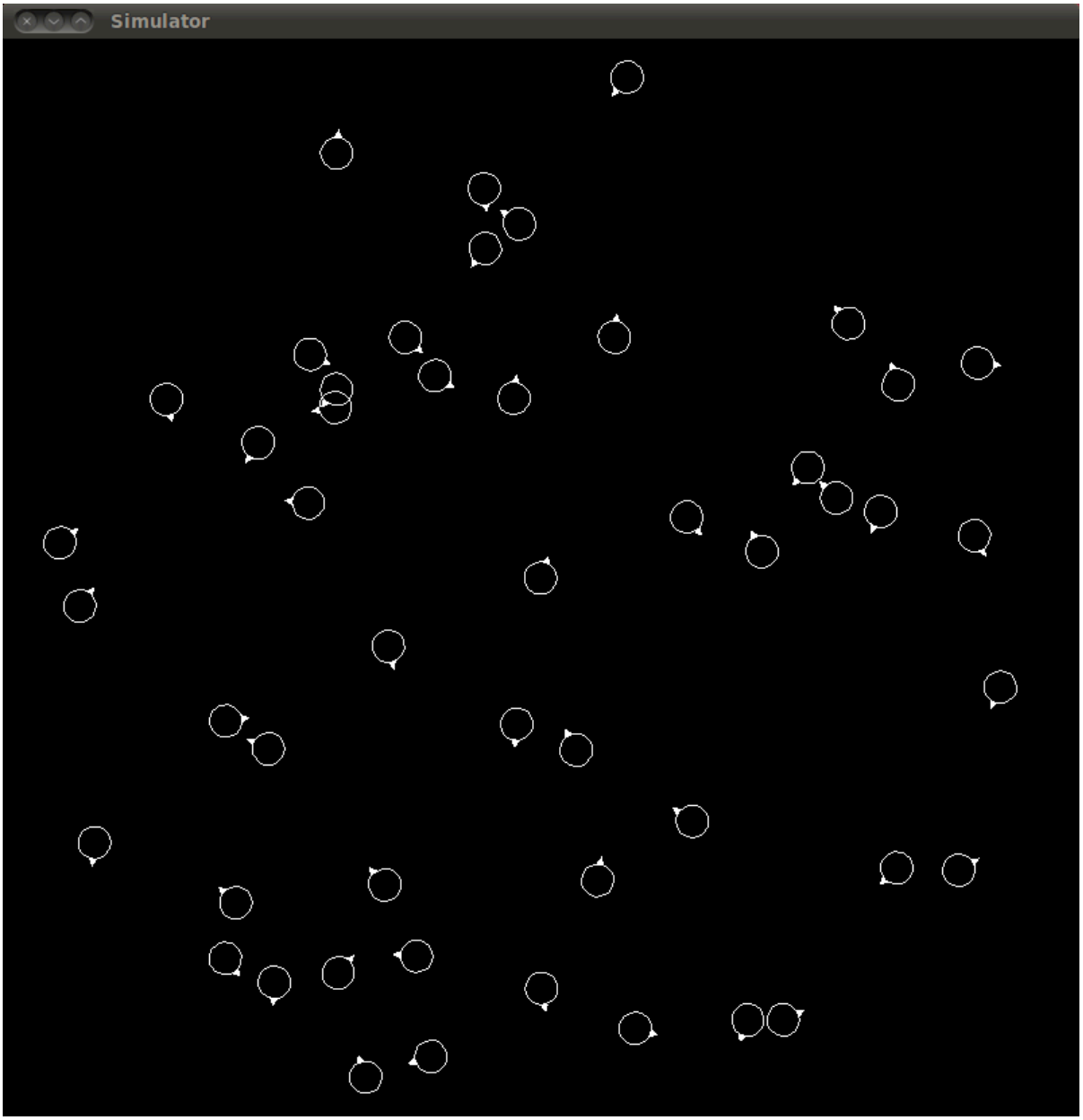


Fig. 28: The Initial State of the Simulator Upon Startup. 50 Unassigned Robots Are Scattered Around the 0,0 Global Coordinate No More Than a Distance of 1.0 From the Center, Located at the Center of the Window.

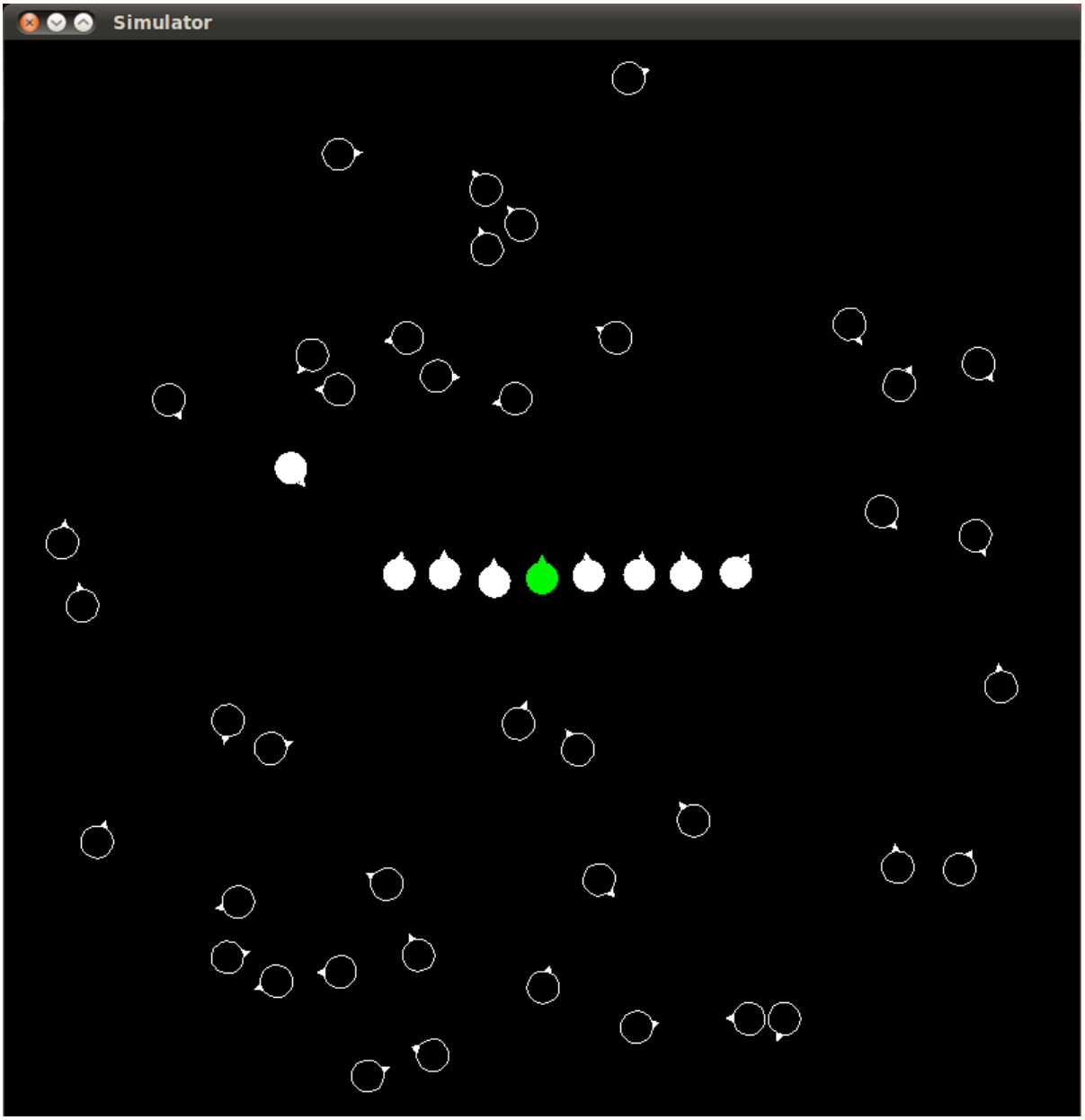


Fig. 29: A Formation Phase Transitioning Via the Push Auction Algorithm.

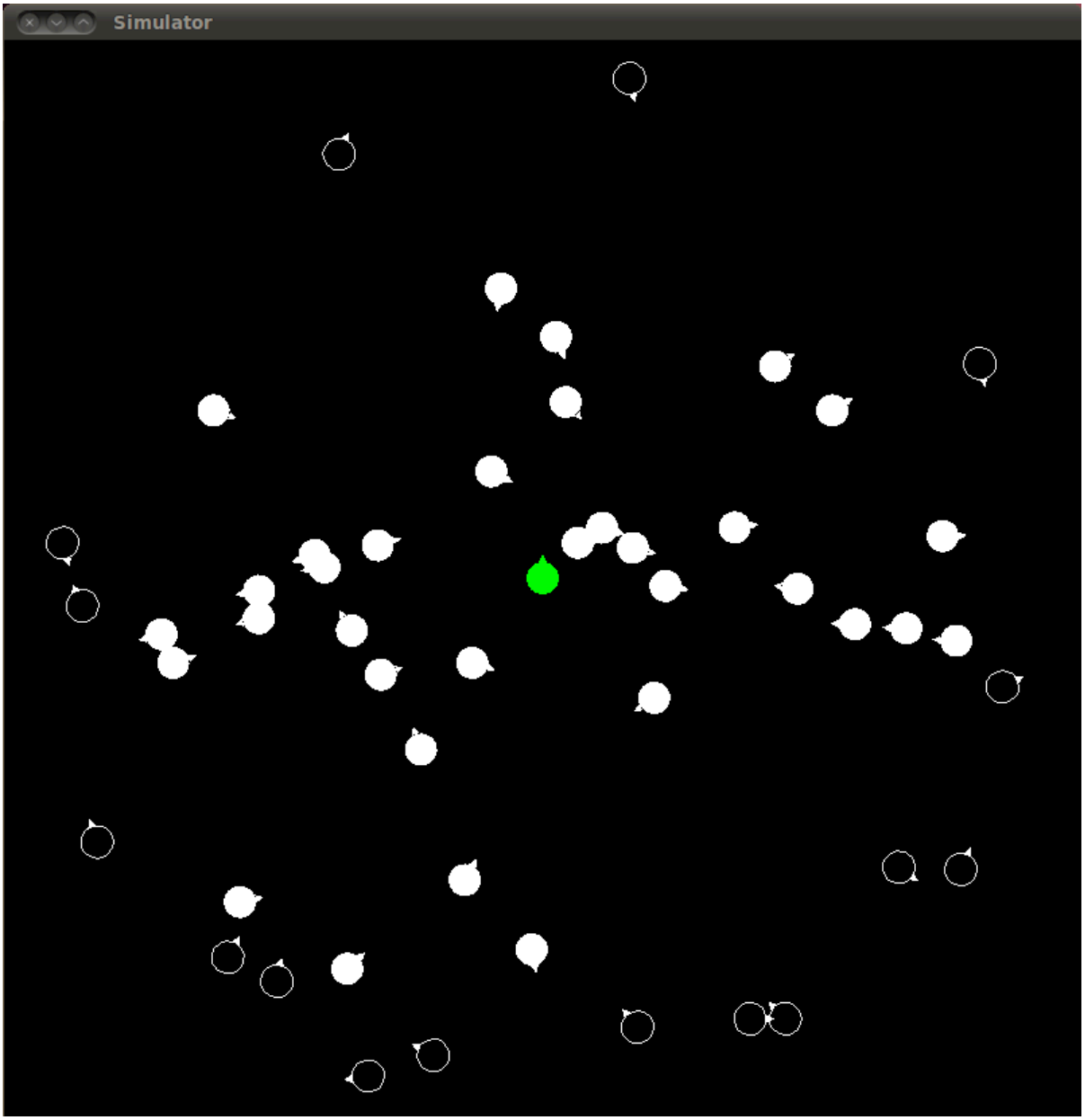


Fig. 30: A Phase Transition in Progress Using the Insertion Auction Algorithm.

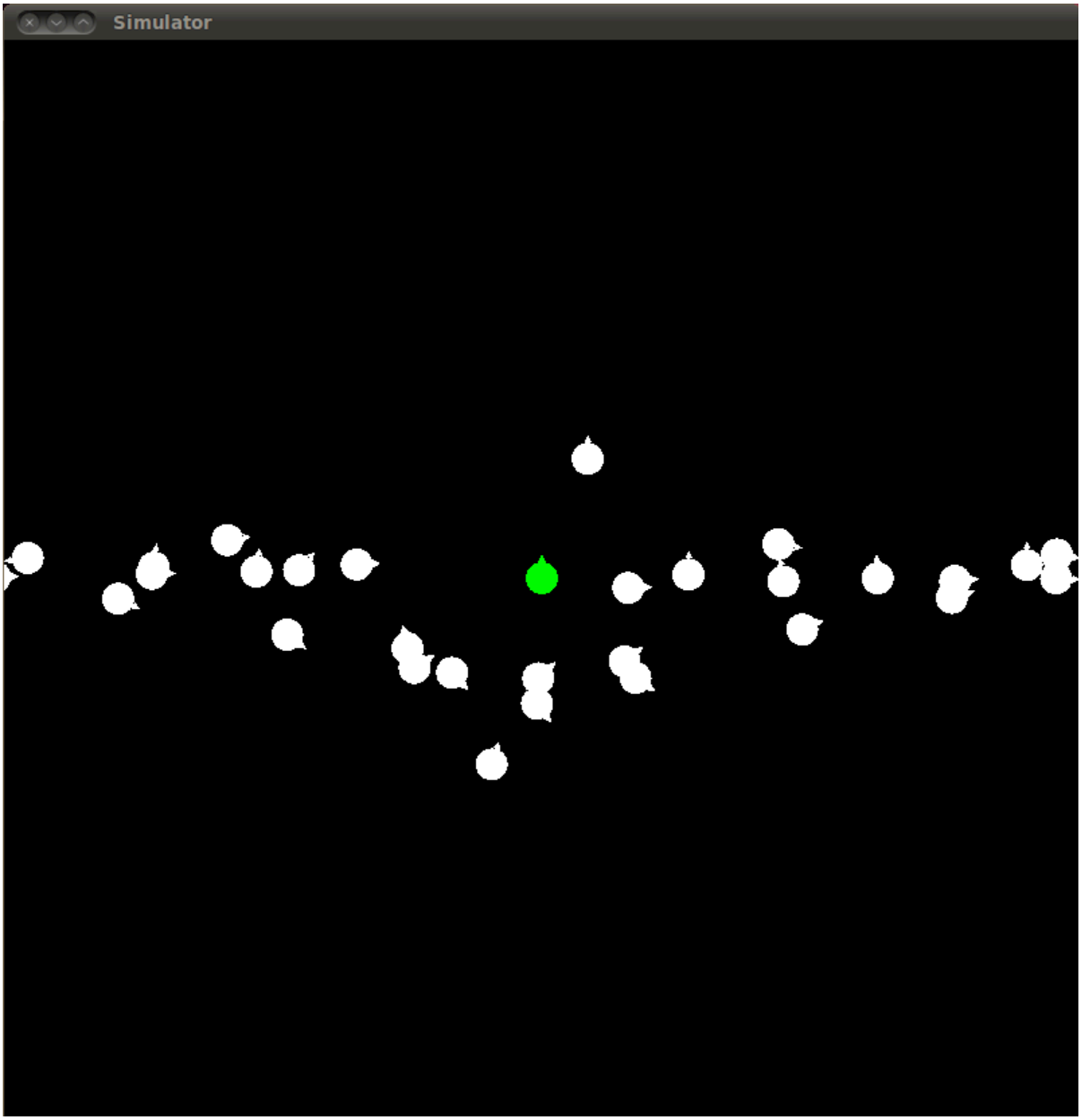


Fig. 31: A Phase Transition Using the Insertion Auction, Near the Convergence Point.

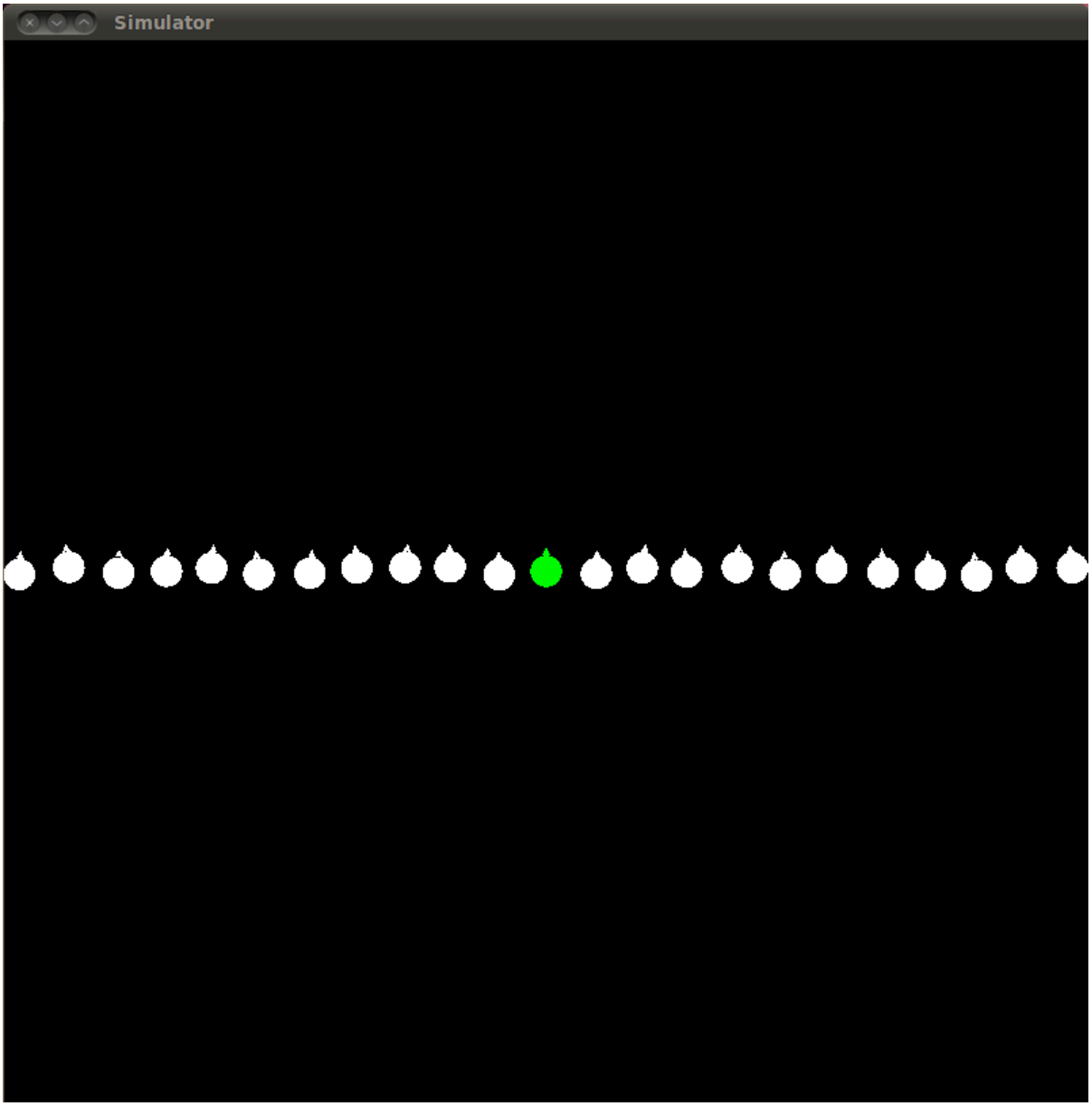


Fig. 32: A Fully Phase Transitioned Formation, Converged at a Linear Formation.

Restrictions

There are several restrictions on the two auction methods that have been adopted for this implementation in order to limit the scope of this work. While the

algorithms remain as general as possible, this implementation has focused on showing the differences between the two auction methods.

The CATALST [19] control architecture has support for multi-function formations. Formations that are defined by multiple functions greatly increase the complexity of the proposed auction algorithms. However, this work is currently limited to single-function formations.

This implementation of the push auction algorithm has been limited to one auction at a time per cell. In a single-function formation the conditions in which a cell may have the opportunity to hold simultaneous auctions is limited to a formation with one cell, the seed c_{seed}. This restriction limits the possibility of confusion over which bids were transmitted for which auction.

The agents in this simulation are treated as zero dimensional points and have no provision for obstacle avoidance or collision detection. This was restriction results in much simpler path planning for the agents; something that is beyond the scope of this work.

CHAPTER V

RESULTS AND ANALYSIS

Results

The method used to evaluate the two distributed auction algorithms begins with generating randomly chosen starting positions for the initial unassigned robots. This is accomplished by means of a random point generator program written in c++ and compiled in GCC. The program is run in a loop where it generates two independent random floating-point numbers between -1.0 and 1.0. This provides an (x,y) pair that is then tested for distance to (0,0), the center of the viewport. If the distance exceeds 1.0, then the pair is thrown out and the loop comes around again. If the pair is within a distance of 1.0 to (0,0) the pair are added to an output file. The first pair generated is (0.0,0.0) for each run of the random point generator. In this fashion 1000 pairs were generated and placed into a file. Thirty of these files were generated. These files were then used to seed initial robot locations in trial runs of the simulator. The reason for including the (0.0,0.0) point in each set of random starting locations was to control for differences in starting location of c_{seed}.

A trial run of the simulator consists of a command line invocation of the simulator binary executable with the appropriate parameters passed. This process was automated by means of the Perl script mentioned in Chapter IV. The pipeline script contains a method of addressing the 30 random point files, which it

systematically feeds to the simulator. A second Perl script was created in order to parse the various results of each run into a digest for compilation into tables.

The parameters iterated on by the pipeline were formation function, number of robots, auction method, and trial number. Formation function was limited to four options: linear, chevron, parabolic, and sine wave. There were three options for number of robots: 10, 50, and 100 robots. The auction method parameter was varied between push method and insertion method auctions.

Two additional auction methods were added as a control for the trials. The first, random push, is identical to the push auction method in every way save for the selection of the winning bidder. Rather than sort the bids for the lowest, a random number is chosen from 0 to the highest index of the received bid queue. The bid located at the index indicated by that random number is chosen as the winner. Along the total distance traveled parameters, this method replicates the behavior of the pre-assigned neighbor method.

The second additional auction method, random insertion, was created as a control for the insertion method. This auction method is the same as the insertion method except for the winning bid selection. As with the random push method, the random insertion method has an auctioneer randomly select a winner rather than finding the lowest bid.

A total of 1440 trial runs were performed, that being the product of 30

trials across 4 methods, 4 formation functions [Fig. 33], and 3 initial amounts of robots: 10, 50, 100.

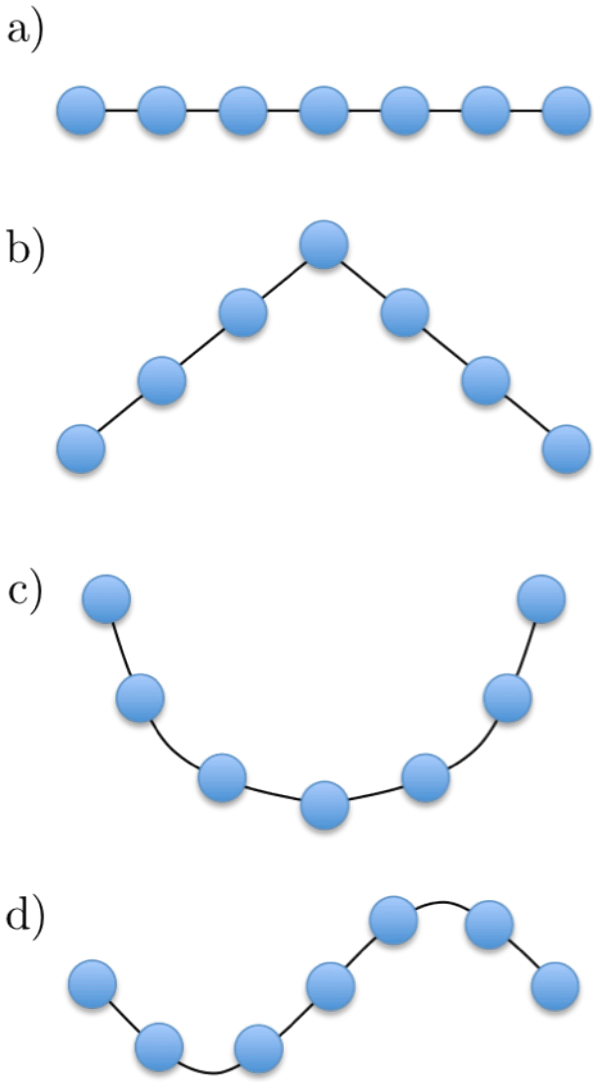


Fig. 33: Formation Functions Chosen For Trial Runs **a)** f(x)=0 **b)** f(x)=-|x| **c)** f(x)=x^2 **d)** f(x)=sin(x)

The results generated by the 1440 trial runs of the simulator were averaged across the 30 trials per configuration of formation function, auction method, and number of robots. Tables 1, 2, and 3 show each of those 48

configurations described in terms of total time to converge, average time to converge per agent, total distance traveled, average distance traveled per agent, total messages sent, average messages sent per agent, and messages sent per turn.

Robots	Auction Type	Total Time to Converge	Avg. Time to Converge/Agent
10	Insertion	135.18	6.27
	Random Insertion	139.11	6.27
	Push	397.39	89.35
	Random Push	386.07	123.63
50	Insertion	383.61	33.26
	Random Insertion	440.36	33.58
	Push	2786.62	718.25
	Random Push	3629.89	1320.88
100	Insertion	764.53	67.82
	Random Insertion	841.18	68.03
	Push	10695.04	2884.51
	Random Push	12788.04	4398.83

Table 1: Trial Runs, Total Time to Converge and Avg. Time to Converge Per Agent, Both Measured In Time Steps

Robots	Auction Type	Total Distance Traveled	Avg. Distance Traveled/Agent
10	Insertion	6.57	0.66
	Random Insertion	6.70	0.67
	Push	5.20	0.52
	Random Push	5.99	0.60
50	Insertion	77.44	1.55
	Random Insertion	85.43	1.71
	Push	45.84	0.92
	Random Push	62.13	1.24
100	Insertion	284.63	2.85
	Random Insertion	306.39	3.06
	Push	185.78	1.86
	Random Push	224.19	2.24

Table 2: Trial Runs, Total Distance Traveled and Avg. Distance Traveled Per Agent, Both Measured In Terms of 33.33 Robot Radii Per Distance Unit

Robots	Auction Type	Total Messages Sent	Avg. Messages Sent / Agent	Avg. Messages Sent / Step
10	Insertion	2373.57	237.36	17.61
	Random Ins.	2445.63	244.56	17.63
	Push	5420.32	533.60	13.59
	Random Push	4533.15	453.32	11.75
50	Insertion	35713.19	714.26	93.17
	Random Ins.	41249.09	824.98	93.73
	Push	202608.48	4046.96	72.67
	Random Push	224941.66	4498.83	61.94
100	Insertion	143427.59	1434.28	187.63
	Random Ins.	158596.99	1585.97	188.63
	Push	1545750.11	15457.50	144.53
	Random Push	1657400.65	16574.01	129.57

Table 3: Trial Runs, Total Messages Sent, Avg. Messages Sent Per Agent, Avg. Messages Sent Per Step

It is also insightful to see data from a particular trial run over the course of the run. The data presented in [Fig. 32] are from two simulator trial runs. One is from a push method run with 50 robots in a linear formation and the other is from an insertion method run with 50 robots in a linear formation. The x-axis measures time steps and the y-axis measures total error in the formation. The total error for the formation is determined by summing the magnitudes of the translational error vectors at each time step. The area under the graph for each run represents the total distance traveled [Fig. 32].

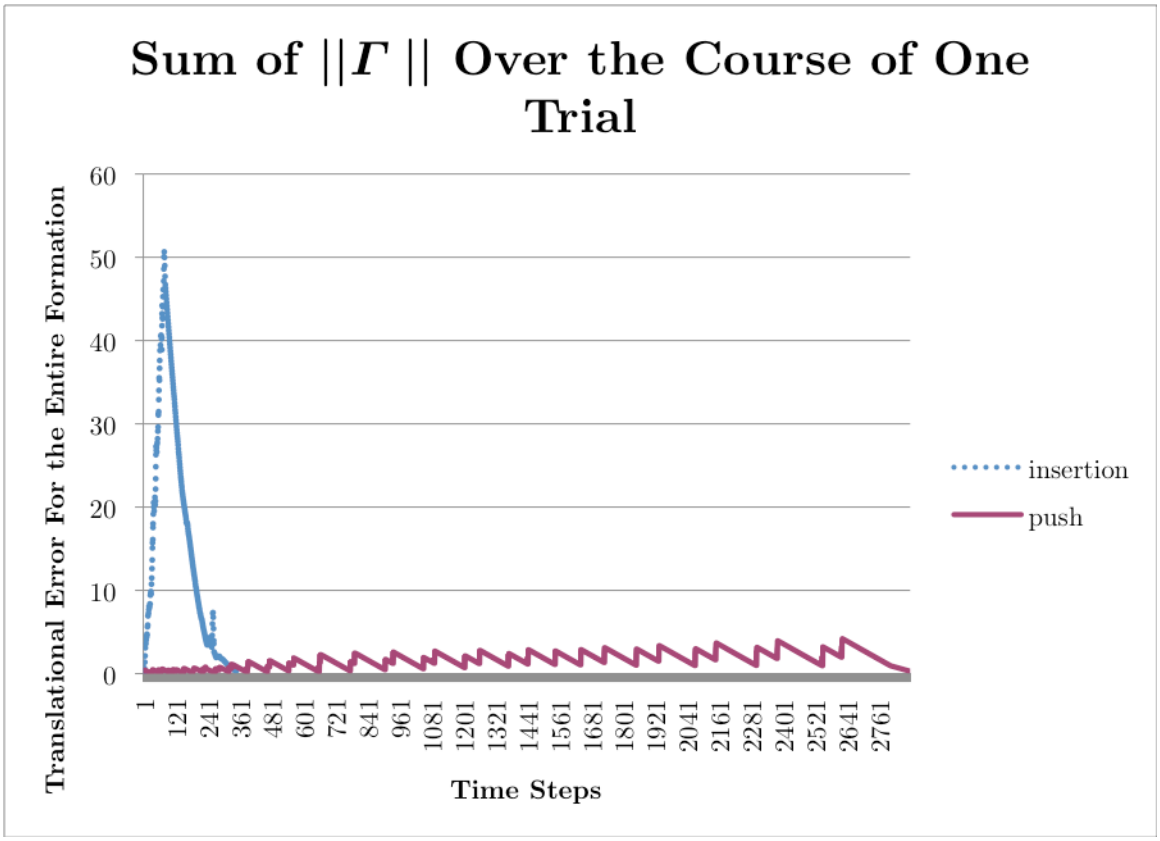


Fig. 34: Total Error in the Formation for Push (solid) and Insertion (dotted).

Analysis

The results showed a significant divergence between the two methods as implemented. The first parameter of comparison is total time to converge. This is the amount of time steps from the transmission of the formation definition to the seed cell until the period of *quiescence*. For the purpose of evaluating this project, the term *quiescence* is applied to the condition in which there are no unassigned robots and the sum of the magnitudes of the translational error vectors of all cells in the formation is near zero for five consecutive time steps. Once *quiescence* is reached,

the simulator automatically stops itself and dumps the gathered data.

The total time to converge for the insertion method was consistently quicker than the push method. This can be shown in the ratio of the time steps to converge for insertion vs. push across 10, 50, and 100 robots: 0.34, 0.14, and 0.07. It is clear that as the number of robots increases, the parallel nature of the insertion method dominates the two-by-two nature of the push method. The average time to converge tells a similar story with the ratios of average time to converge for insertion to push for 10, 50, and 100 robots being 0.06, 0.05, and 0.02. Again, the performance of the insertion method with respect to the push method only improves as more robots are added and the parallel nature of the insertion method is exploited further.

The total distance traveled metric tells a different story, however. The ratios of total distance traveled for insertion vs. push method for 10, 50, and 100 robots were 1.26, 1.69, and 1.53. This implies that the push method is superior at minimizing the total distance traveled. For many applications, this factor will be more important than the total time to converge, since it represents energy costs. For space based solar power satellites, this factor alone is enough to recommend the push method over the insertion method. Satellites have limited fuel supply, and as such, minimizing the overall distance traveled is vastly more important than time to converge.

The final metric to be considered is that of messages sent per time step.

While data were collected for total messages sent and average messages sent, the measure of messages sent per time step is a more accurate view of the congestion caused by messaging during the phase transition process. The ratios of messages sent per time step for insertion vs. push method on 10, 50, and 100 robots were 1.50, 1.28, and 1.30. By this measure, the push method uses fewer messages per time step than the insertion method.

The results of comparisons between the two auction methods and their random counterparts illustrate a few useful points. The 10 robot random trials tended to be indistinguishable from the push and insertion method auctions. Across the 50 and 100 robot trials, the benefit from the auction methods above randomly choosing bids ranged from 10-25%.

Algorithm Analysis

In the analysis of the two auction algorithms, the variable n is assigned the number of robots. Since any computations done by individual agents will be dominated by the computation of the overall system, the computation of agents is not considered in this analysis. The key aspect of this analysis is the number of messages required to obtain a result.

Push Auction Method

In order to find the complexity of messaging in the push auction method with respect to the number of robots, the $O(n)$, the example from Chapter III will

help to illustrate the generalized approach. The example begins with a total of n agents; one of which has been designated as c_{seed}. Randomly scattered robots (n-1) surround this formation of one. At this point c_{seed} will hold two auctions. The first auction will produce one message for the auction announcement, and worst case of (n-1) bids, and then one message to announce the winner of the auction. This produces (n-1+2) or (n+1) messages. The second auction, which will be held by c_{seed}, can be thought of in the same terms as the first, since there are no guarantees in the order of auctions. Thus, we say that the two auctions within a radius of 1 from c_{seed} require, in the worst case, 2(n+1) messages. Each of the two cells now neighboring c_{seed} will hold auctions that will have a worst case of 2(n-1)-2 messages. The final term (-2) represents two fewer bids received from the unassigned robots, which now number (n-3). When considered across the entire operation of the algorithm, the description of the number of messages is calculated as $(n - 1)[(n - 1) - (n / 2)]$. The first $(n - 1)$ factor represents the total number of auctions. The second factor represents the number of bids received, in the worst case. The $(n / 2)$ term represents the worst-case number of bids for each half of the formation. This term assumes that each formation will be balanced. It is conceivable that n/2 agents could join the formation on one side of c_{seed} before a single agent joins the other side.

When $(n - 1)[(n - 1) - (n / 2)]$ is simplified, the result is $(n^2/2)-(3n/2)+1$. So in the final analysis of the push auction method, the (n^2) term dominates, and the

complexity of messaging in order to organize n robots into a formation is determined to be $O(n) = n^2$.

Insertion Auction Method

The analysis for the insertion method is is quite similar to the push method. Again, there are n agents to begin the scenario. One has been designated as c_{seed}. At this point, the remaining (n-1) unassigned robots will each announce an auction, generating (n-1) messages. As c_{seed} receives these messages, it determines which is nearest and responds with a bid, generating one message. The auctioneer, an unassigned robot, will then announce c_{seed} as the winner, generating one more message. The insertion auction held will generate (n+1) messages. As the number of cells increases, the number of auction announcements will decrease, but the number of bids will increase. Thus, the total number of messages in order to converge n agents is determined by (n-1)(n+1) or (n^2-1). Again, since the n^2 term dominates the constant term, it is said the complexity of the insertion method with respect to messaging is $O(n) = n^2$.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

Conclusions

The data collected support several conclusions. The primary conclusion of this work is to distinguish the benefits and uses of both the push method and the insertion method for initializing formations of mobile robots.

The data support the conclusion that both auction methods are generalizable to many and varied formation functions. The trials tested four formation functions: $f(x)=0$, $f(x)=-|x|$, $f(x)=x^2$, $f(x)=\sin(x)$. Each one of these performed similarly under all trial runs. While it was observed that $f(x)=\sin(x)$ consistently converged faster, this difference was no more than 10% faster. It is observed that the sine function provided a shorter total distance to travel for each agent by packing more agents closer to the (0,0) point.

The data support the conclusion that when considering metrics regarding time to converge, the insertion method is superior, and improves its performance over that of the push method as the number of robots in the system is increased. This is a result of the parallel nature of the insertion method. This implies that applications that are time sensitive may wish to consider the insertion method. This may apply exclusively to terrestrial systems where refueling from outside the system is a viable option. Such time sensitive systems may include reconnaissance and urban

search and rescue.

Further, the data support the conclusion that along the metric of efficiency, total distance traveled, the push method is superior in each case. The nature of satellites as self-contained systems in the vacuum of space renders this metric particularly applicable to considerations involving satellite systems. Another consideration for the push method is the two-by-two nature of the initialization process. This may be of concern when applying formation initialization methods to large groups of expensive and or delicate robots. While the insertion method allows for much faster convergence, it places all robots in motion in parallel that has the potential to yield many more robot-to-robot collisions or collision avoidance behavior. While the simulator treats all robots as zero dimensional points, this is obviously not the case in the real world. The robots will need to maneuver to avoid one another as they travel towards their positions in the formation.

Future Work

Communication Model

The topic-based publish/subscribe model is proposed as a means to facilitate communications between groups of cells/unassigned robots. The publish/subscribe paradigm allows a distributed group of agents to send (i.e. publish) and receive (i.e. subscribe) various types of information regarding select topics of interest. This allows state information to be passed among neighbors, while

still allowing broadcast messages to nearby agents. A topic is addressed with a unique identifier, which, for our purposes, is a formation-relative position p_j. We write the topic identified by this formation-relative position as:

$$\mathrm{T}(p_j) = \{\ p_j,\ s_j\} \qquad \text{Eq. 2}$$

This communications model is particularly useful when using multi-function formations. As pointed out by Sujit and Beard [26], there may be cases when two agents wish to auction the same resource. This is especially a problem when those two agents are not neighboring one another and for this reason would not normally have contact with one another. The publish/subscribe model provides a framework upon which to extend the two auction algorithms of this work in order to alleviate some of the issues that arise as a result of having multiple-function formations.

Cells As Bidders in the Push Auction Algorithm

In the push auction method it has been suggested that cells could also serve as bidders. There are two conditions that determine whether cells would proffer bids based on auction announcements received. First, the bidding cell's neighborhood must not be full ($n < n_{max}$). Second, the distance from the formation-relative position of the bidder p_{bidder} to the seed p_{seed}, represented as $||p_{bidder}\text{-}p_{seed}||$, must be less than the distance from the formation-relative position of the position being auctioned p_j to p_{seed}, represented $||p_j\text{–}p_{seed}||$ [18].

In order to properly account for the effects of cells bidding in push auctions, it is reasonable to add a term to the existing bid function. An altered bid function for a cell in a push auction is:

$$\mathrm{B}(p_j) = d + X\ n \qquad \text{Eq. 3}$$

The first term consists of the distance factor *d.* The second term consists of the number of current neighbors n (for an unassigned robot this would always be zero) weighted by a *relation cost modifier* X [18]. The relation cost modifier represents the cost of breaking an existing neighbor relationship. The higher the value of X the harder it would be for a cell to win an auction. There is no conceivable case when a cell could have zero neighbors and bid in an auction because the only occasion when a cell would have zero neighbors is when it is the seed. In this case, the seed would be the only cell in the formation and there could be no auctions that originated from a cell other than itself. Cells are disallowed from bidding in their own auctions.

Energy Cost Modifier

The *energy cost modifier* is a coefficient that modifies the distance factor d in order to account for difficulties in the path of the agent that may not be accounted for in the distance alone [Eq 3].

$$\mathrm{B}(p_j) = E\ d + X\ n \qquad \text{Eq. 3}$$

Such difficulties may include rough terrain that requires a slower approach.

Seed Selection and Phase Transition

In this work, an auction-based method has been implemented in order to facilitate a phase transition of a swarm of robots to a formation of robots. This formation initialization method has been implemented and tested on a subset of the many possible single-dimensional formation definitions, though the phase transition method is applicable to any valid formation definition. The implemented method requires that a seed robot be known. At this point, it does not matter whether the seed is selected by the human operators or by a seed election algorithm run by the robots themselves, though seed choice is expected to have an effect on the overall performance of the swarm to formation phase transition. A method that allows the swarm to autonomously and in a distributed manner select a seed robot, would, it is expected, greatly increase the effectiveness of the auction process. A key goal to consider when implementing a seed election, is finding that robot that is currently located in such a way that it would minimize the time and fuel spent, total distance traveled, during the phase transition.

Multi-Function Formations

The proposed auction algorithms have been evaluated only with respect to single function formations. However, as Mead [19] discussed at length, multi-function formations can be implemented and these would also, conceivably benefit from

auction initialization. The topic-based publish/subscribe communication model could provide an invaluable tool in the effort to coordinate auctions among neighbors wishing to fill the same neighboring formation-relative position.

Among the two auction methods, the insertion method is specifically applicable to multi-function formations. A possible consequence of multi-function formations is that of isolating agents inside the bounds of the formation without including those agents in the formation itself. If the initial swarm/collection were dense enough, it is plausible that a formation could form and leave a pocket of agents such that joining the formation would be made more difficult by being surrounded by agents already in the formation. The insertion auction method, which causes unassigned robots to hold auctions and cells to bid on those auctions, provides a solution to this issue. The nearest cells to the isolated group of unassigned robots would win the auctions and those unassigned robots would join the formation at the nearest formation-relative positions.

Formation Repair

In the event of robot failure, it may be desirable for robots with established positions in the formation to break free and rejoin in a new formation-relative position. Including a term in the weighted sum used to calculate the bid can control this behavior. This term would be a multiplier to weight the number of preexisting relations to be broken if the bid were accepted, as discussed in the *Cells*

As Bidders in the Push Auction Method section above. The formation repair function would likely be initiated upon confirmation of loss of communications with neighbors. The neighbors of the lost robot would then send an auction notice for a robot to fill the lost robot's formation-relative position

Formation Obstacle Avoidance

The phase transition metaphor suggests a method for allowing avoidance of obstacles. As cells at the highest radius from the c_{seed} encounter an obstacle, they may loosen their bonds to their neighbors, similar to solid matter melting into liquid matter. For a widely spaced formation encountering a relatively small obstacle, perhaps something no more than twice as the diameter of the robot, the neighbor bonds may be loosened but not broken, allowing the robots in the path of the obstacle to shuffle out of the way. For tighter formations or larger obstacles, it may be necessary to melt local portions of the formation, breaking or temporarily ignoring relations with neighbor robots in order to flow out of the way of the obstacle. Once the robot determines itself to have passed the influence of the obstacle, it would either re-activate its temporarily ignored relations or begin bidding for open slots in the formation. Robots that have melted or sublimated would position themselves by reverting to Reynold's rules of swarming [10]. If the entire formation must be melted in order to facilitate obstacle avoidance, then the phase transition method may be employed once again to precipitate the formation from the swarm of robots.

REFERENCES

[1] Atay, N., Bayazit, B. (2008), Emergent Task Allocation for Mobile Robots, Robotics: Science and Systems III, MIT Press 2008.

[2] Aung, M et al (2004), An Overview of Formation Flying Technology Development for the Terrestrial Planet Finder Mission, Aerospace Conference, 2004 Proceedings, 2004 IEEE.

[3] Balch, T., Arkin, R. (1998), Behavior Based Formation Control for Multirobot Teams, IEEE Transactions on Robotics and Automation, Vol.14, No.6, December 1998

[4] Beer, B., Mead, R., Weinberg, J.B. (2010). A Distributed Method for Evaluating Properties of a Robot Formation.

[5] Bekey, G., Bekey, I., Criswell, D., Friedman, G., Greenwood, D., Miller, D., & Will, P. (2000). Final Report of the NSF-NASA Workshop on Autonomous Construction and Manufacturing for Space Electrical Power Systems. 4-7 April, Arlington, Virginia.

[6] Cheng, J., Cheng, W., & Nagpal, R.(2005), Robust and Self-repairing Formation Control for Swarms of Mobile Agents. In Proceedings of AAAI-05, 59-64. Pittsburgh, Pennsylvania.

[7] Cunha, R., Silva, A., Loureiro, A., Ruiz, L. (2005), Simulating Large Wireless Sensor Networks Using Cellular Automata, Proceedings of the 38th Annual Simulation Symposium, ANSS 2005

[8] Dudek, G., Jenkin, M., Milios, E., A Taxonomy of Multirobot Systems. "Robot Teams: From Diversity to Polymorphism" edited by Tucker Balch and Lynne Parker, 2002 A.K. Peters, Ltd.

[9] ESA Darwin Flotilla (2002), Illustration retrieved from

[10] Fredslund, J, & Mataric, M.J. (2002), Robots in Formation Using Local Information. The 7th International Conference on Intelligent Autonomous Systems, Marina Del Rey, California.

[11] Gerkey, B.P., Mataric, M.J. (2002), "SOLD!: Auction Methods for

Multirobot Coordination." Robotics and Automation, IEEE Transactions on, Vol 18 Issue 5, Oct. 2002, 758-768.

[12] Gradwell, P., & Padget, J. (2007), "A Comparison of Distributed and Centralized Agent Based Bundling Systems," ICEC '07, August 19-22, 2007 Minneapolis, Minnesota, USA.

[13] Heil, R.(2004), "A Trilaterative Localization System For Small Mobile Robots in Swarms." A Thesis submitted to the department of Electrical and Computer Engineering and the Graduate School of the University of Wyoming. Laramie, Wyoming, August 2004.

[14] Kishimoto, A., Sturtevant, N., (2008) "Optimized Algorithms for Multi-Agent Routing." Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Padgham, Parkes, Muller and Parsons(eds.), May, 12-16, 2008, Estoril, Portugal, pp. 1585-1588.

[15] Lagoudakis, M. G., Berhault, M., Koenig, S., Keskinocak, P., Kleywegt, A., (2004), "Simple Auctions with Performance Guarantees for Multi-Robot Task Allocation." Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sept. 28 – Oct. 2, 2004, Sendai, Japan.

[16] Lemay, M., Michaud, F., Letourneau, D., Valin, J., Autonomous Initialization of Robot Formations.

[17] Lewis, M., Tan, K., High Precision Formation Control of Mobile Robots Using Virtual Structures. Autonomous Robots 4, 387-403 (1997) Kluwer Academic Publishers, The Netherlands.

[18] Long, R., Mead, R., & Weinberg J.B. (2010). Distributed Auction-Based Initialization of Mobile Robot Formations. Proceedings of the Twenty Fourth AAAI Conference on Artificial Intelligence, Atlanta, Georgia, USA, July 11-15, 2010. pp 1949-1950.

[19] Mead, R (2008), Cellular Automata for Control and Interactions of Large Formations of Robots, A Thesis Submitted in Partial Fulfillment of the Requirements for the Master of Science Degree, Southern Illinois University at Edwardsville, Edwardsville, Illinois.

[20] Mead, R, Weinberg, J., Croxell, J. (2007), A Demonstration of a Robot Formation Control Algorithm and Platform, 2007, AAAI

[21] Oung, R., Bourgalt, F., Donovan, M., D'Andrea, R.(2010), "The Distributed Flight Array." Proceedings of the 2010 IEEE Conference on Robotics and Automation, Anchorage, Alaska, USA, May 3-8, 2010. pp. 601-607.

[22] Reynolds, C.W. (1987), Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics, SIGGRAPH '87 Conference Proceedings, 21(4),25-35

[23] Sawada, Y., Tsubouchi, T.(2010), "Autonomous Re-alignment of Multiple Table Robots." Proceedings of the 2010 IEEE Conference on Robotics and Automation, Anchorage, Alaska, USA, May 3-8, 2010. pp. 1098-1099.

[24] Smith, R., "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computers, Vol C-29, No. 12, 1980.

[25] Spears, W., Heil, R., Spears, D., Zarzhitsky, D., Physicomimetics for Mobile Robot Formation. AAMAS '04, July19-23, 2004, New York , New York, USA.

[26] Sujit, P.B., & Beard, R.(2007), "Distributed Sequential Auctions for Multiple UAV Task Allocation." Proceedings of the 2007 American Control Conference, New York City, New York, USA, July 11-13, 2007.

[27] Suzuki, I., Yamashita, M., Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. 1999 Society for Industrial and Applied Mathematics SIAM J. COMPUT. Vol. 28, No.4, pp. 1347-1363.

[28] Yamaguchi, H., Arai, T., & Beni, G. (2001)," A Distributed Control Scheme for Multiple Robotic Vehicles to Make Group Formation", Robotics And Autonomous Systems, (36), 125-147

[29] Yu, C., Nagpal, R.(2010), "Biologically Inspired Control for Multi-Agent Self-Adaptive Tasks." Proceedings of the Twenty Fourth AAAI Conference on Artificial Intelligence, Atlanta, Georgia, USA, July 11-15, 2010. pp. 1702-1707.

APPENDIX A

DATA FROM TRIAL RUNS

Robots	Auction Type	Formation Shape	Total Time to Converge	Avg. Time to Converge
10	insertion	Line	135.55	6.28
		Chevron	136.17	6.27
		Parabola	135.73	6.27
		Sine	133.27	6.27
	random insertion	Line	142.79	6.28
		Chevron	134.60	6.27
		Parabola	139.03	6.27
		Sine	140.00	6.27
	push	Line	348.40	89.47
		Chevron	349.57	89.80
		Parabola	432.93	89.13
		Sine	458.67	89.00
	random push	Line	384.30	125.20
		Chevron	382.47	121.57
		Parabola	393.03	121.83
		Sine	384.47	125.93
50	insertion	Line	393.45	33.24
		Chevron	405.00	33.40
		Parabola	385.87	33.23
		Sine	350.13	33.17
	random insertion	Line	448.69	33.59
		Chevron	466.17	33.53
		Parabola	455.60	33.57
		Sine	391.00	33.63
	push	Line	2818.40	753.67
		Chevron	2868.50	750.03
		Parabola	3086.70	744.30
		Sine	2372.87	625.00
	random push	Line	3766.23	1348.17
		Chevron	3799.57	1371.40
		Parabola	3710.10	1348.70
		Sine	3243.67	1215.23
100	insertion	Line	808.52	67.76
		Chevron	775.17	67.83
		Parabola	784.13	67.97
		Sine	690.30	67.73
	random Insertion	Line	851.72	68.07
		Chevron	849.40	68.13
		Parabola	869.57	68.03
		Sine	794.03	67.90
	push	Line	11014.30	3026.40
		Chevron	11216.44	3052.56
		Parabola	11668.70	3069.80
		Sine	8880.70	2389.30
	random push	Line	13417.07	4591.30
		Chevron	13344.37	4559.17
		Parabola	12990.17	4462.43
		Sine	11400.57	3982.43

Robots	Auction Type	Formation Shape	Total Distance Traveled	Avg. Distance Traveled
10	insertion	Line	6.61	0.66
		Chevron	6.65	0.67
		Parabola	6.60	0.66
		Sine	6.40	0.64
	random insertion	Line	6.82	0.68
		Chevron	6.63	0.66
		Parabola	6.61	0.66
		Sine	6.76	0.68
	push	Line	5.16	0.52
		Chevron	5.20	0.52
		Parabola	5.27	0.53
		Sine	5.16	0.51
	random push	Line	6.01	0.60
		Chevron	5.96	0.60
		Parabola	6.05	0.61
		Sine	5.94	0.59
50	insertion	Line	77.91	1.56
		Chevron	81.71	1.63
		Parabola	78.06	1.56
		Sine	72.09	1.44
	random insertion	Line	86.45	1.73
		Chevron	90.63	1.81
		Parabola	88.48	1.77
		Sine	76.18	1.52
	push	Line	46.76	0.93
		Chevron	48.51	0.97
		Parabola	50.08	1.00
		Sine	38.01	0.76
	random push	Line	64.69	1.29
		Chevron	65.17	1.30
		Parabola	63.16	1.26
		Sine	55.50	1.11
100	insertion	Line	298.34	2.98
		Chevron	287.98	2.88
		Parabola	296.25	2.96
		Sine	255.97	2.56
	random Insertion	Line	314.21	3.14
		Chevron	306.83	3.07
		Parabola	316.53	3.17
		Sine	287.97	2.88
	push	Line	191.62	1.92
		Chevron	196.20	1.96
		Parabola	201.45	2.01
		Sine	153.85	1.54
	random push	Line	235.68	2.36
		Chevron	234.54	2.35
		Parabola	226.93	2.27
		Sine	199.60	2.00

Robots	Auction Type	Formation Shape	Total Messages Sent	Avg. Messages Sent	Avg. Messages Per Step
10	insertion	Line	2380.14	238.01	17.63
		Chevron	2391.40	239.14	17.58
		Parabola	2383.57	238.36	17.66
		Sine	2339.17	233.92	17.59
	random insertion	Line	2511.97	251.20	17.69
		Chevron	2364.70	236.47	17.65
		Parabola	2444.23	244.42	17.58
		Sine	2461.63	246.16	17.58
	push	Line	4537.10	453.71	13.04
		Chevron	4551.00	455.10	13.04
		Parabola	6065.87	592.66	14.04
		Sine	6527.30	632.92	14.25
	random push	Line	4469.57	446.96	11.64
		Chevron	4510.47	451.05	11.81
		Parabola	4694.37	469.44	11.94
		Sine	4458.20	445.82	11.61
50	insertion	Line	36682.34	733.65	93.34
		Chevron	37803.60	756.07	93.34
		Parabola	35937.97	718.76	93.35
		Sine	32428.83	648.58	92.65
	random insertion	Line	42068.55	841.37	93.90
		Chevron	43777.27	875.55	93.94
		Parabola	42735.67	854.71	93.92
		Sine	36414.87	728.30	93.13
	push	Line	202222.67	4040.90	71.76
		Chevron	207381.73	4147.63	72.31
		Parabola	229463.73	4579.37	74.36
		Sine	171365.80	3419.95	72.25
	random push	Line	235579.27	4711.59	62.55
		Chevron	236512.47	4730.25	62.26
		Parabola	230019.20	4600.38	62.00
		Sine	197655.70	3953.11	60.95
100	insertion	Line	152120.97	1521.21	188.27
		Chevron	145524.97	1455.25	187.77
		Parabola	147314.77	1473.15	187.90
		Sine	128749.67	1287.50	186.59
	random Insertion	Line	160641.38	1606.41	188.77
		Chevron	160238.47	1602.38	188.74
		Parabola	164216.43	1642.16	188.97
		Sine	149291.67	1492.92	188.02
	push	Line	1580608.50	15806.07	143.51
		Chevron	1615358.44	16153.59	144.02
		Parabola	1701480.20	17014.81	145.82
		Sine	1285553.30	12855.52	144.77
	random push	Line	1743463.53	17434.64	129.94
		Chevron	1735497.70	17354.99	130.06
		Parabola	1684675.50	16846.76	129.69
		Sine	1465965.87	14659.66	128.59

APPENDIX B

CODE

randc.cpp

```
//This c++ program generates random x,y coordinates
//for use in seeding the simulator

#include <fstream>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string>
#include <cmath>

using namespace std;

float frand(const float min = 0.0f, const float max = 1.0f)
{
    return min + (max - min) * float(rand()) / float(RAND_MAX);
}

bool distance_ok(float x, float y)
{
   bool answer = false;
   float thing = sqrt((x*x)+(y*y));
   answer = thing <= 1.0;
   return answer;
}


int get_sign()
{
   int answer;
   if(frand()>0.5)
   {
      answer = 1;
   } else {
      answer = -1;
   }
   return answer;
}

int main(int argc, char * argv[])
{
   int num = atoi(argv[1]);
   string output = argv[2];
   srand(time(NULL));
   ofstream stuff;
   stuff.open(output.c_str());
```

```
    for(int i=0;i<num;i++)
    {
        int sign;
        if(frand()>0.5)
        {
            sign = 1;
        } else {
            sign = -1;
        }
        if(i<2)
        {
            sign = 0;
        }
        float x=2.0,y=2.0;
        if(i==0)
        {
            x = y = 0.0;
        } else {
            while(!distance_ok(x,y))
            {
                x = frand()*get_sign();
                y = frand()*get_sign();
            }
        }
        stuff << x << endl;
        stuff << y << endl;
    }
    stuff.close();
    sleep(2);
    return 0;
}
```

runSim.pl
//This Perl script makes up the trial run pipeline

```
#!/usr/bin/perl
use File::Copy;

my $simulator = "~/git/simulator/Simulator";
my $seed_location = "~/git/simulator/support/seeds/random_xy_seeds_";
my $storage_location = "~/git/simulator/data";
my $output_dir = "~/git/simulator";
my $stdout_file = " > " . $output_dir . "/stdout.out";
my $trials = 30;

#save_data(1);

my @formation_types = (0,4,6,9);

my @nums_of_robots = (10,50,100);

my @push_or_insertion = ('push','insertion', 'rpush','rinsertion');

my $total_runs = $trials * scalar(@formation_types) * scalar(@num_robots);

print "Beginning a run of $total_runs trials.\n";

for my $seed (1 .. $trials){
   for my $formation_type (@formation_types){
      for my $num_robots (@nums_of_robots){
         for my $auction_type (@push_or_insertion){
            my $type = '';
            if ($auction_type eq 'insertion'){
               $type = "-i -e 99999";
            } elsif ($auction_type eq 'rpush'){
               $type = "-rpush";
            } elsif ($auction_type eq 'rinsertion'){
               $type = "-rinsertion"
            }
            my $sim_call = $simulator . " -t 1 -s " . $seed_location . $seed . ".txt -f
". $formation_type . " -n " . $num_robots . " " . $type . $stdout_file;
            print "Calling sim on trial $seed \n";
            print "using: $sim_call\n";
            unless(system($sim_call)==0){
```

```
                print "sim_call = " . $sim_call . "\n";
                die "sim did not exit properly.";
            }
            save_data($seed,$formation_type,$num_robots,$auction_type);
        }
    }
  }
}

sub save_data {
   my @stuff = @_;
   $seed = $stuff[0];
   $formation_type = $stuff[1];
   $num_robots = $stuff[2];
   $auction_type = $stuff[3];
   print "Storing data for trial " .$seed . ".\n";
   my $storage_dir = $storage_location . "/".$auction_type."_run_" . $seed .
"_formation-type_" . $formation_type . "_num-robots_". $num_robots;
   unless(not -d $storage_dir){
      die "storage_dir already exists at " . $storage_dir;
   }
   unless(mkdir($storage_dir)){
      die "mkdir failed... trying " . $storage_dir;
   }
   my $mv = "mv *.out " . $storage_dir;
   unless(system($mv)==0){
      print "copy command was ".$mv."\n";
      die "data failed to copy for trial ".$seed;
   }
}
```