2008

Autonomous construction agents: An investigative framework for large sensor network self-management

Joshua Bruce Koch
Iowa State University

Autonomous construction agents: An investigative framework for large sensor network self-management

by

Joshua Bruce Koch

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Mechanical Engineering

Program of Study Committee:
Kenneth M. Bryden, Major Professor
Terrence R. Meyer
Carolyn D. Heising

Iowa State University

Ames, Iowa

2008

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Recent technological advances have made it cost effective to utilize massive, heterogeneous sensor networks. To gain appreciable value from these informational systems, there must be a control scheme that coordinates information flow to produce meaningful results. This paper will focus on tools developed to manage the coordination of autonomous construction agents using stigmergy, in which a set of basic low-level rules are implemented through various environmental cues. Using VE-Suite, an open-source virtual engineering software package, an interactive environment is created to explore various informational configurations for the construction problem. A simple test case is developed within the framework, and construction times are analyzed for possible functional relationships pertaining to performance of a particular set of parameters and a given control process. Initial experiments for the test case show sensor saturation occurs relatively quickly with 5-7 sensors, and construction time is generally independent of sensor range except for small numbers of sensors. Further experiments using this framework are needed to define other aspects of sensor performance. These trends can then be used to help decide what kinds of sensing capabilities are required to simultaneously achieve the most cost-effective solution and provide the required value of information when applied to the development of real world sensor applications.

CHAPTER 1. INTRODUCTION

Developments in microelectromechanical systems, high-speed microprocessors, low-voltage logic devices, and high-capacity memories have encouraged the production of small, low-cost, multifunctional sensor nodes that create microsystems to sense and control an environment. These sensor nodes generally consist of a processor, external memory, receiver/transceiver, power source, and one or more sensors and are capable of processing data, gathering various types of sensory information, and communicating with other nodes in a network. Their small size and low cost makes it feasible to deploy vast numbers of sensors that can provide large amounts of information. However, there are considerable challenges in coordinating and processing this information.

At first glance, it seems logical that a large number of sensors will give a more reliable and relevant picture of the sensed environment, but there are problems associated with processing ever-increasing bits of data. Given massive numbers of sensor devices, how will data, be efficiently and effectively merged and processed to effectuate real-time outcomes? What kinds of sensors and capabilities are required? How many sensors need to be deployed and at what location? The real value in information production comes in refining what is useful to the end user. If a quantity of information is no longer useful, then it is not cost effective to increase sensing resolution for the system. Ultimately, these questions will be determined by the problem trying to be solved and the consequences of not making an optimal decision.

Traditionally, sensors have been administered with a top-down approach in which information is sent from each sensor to a central control unit where data fusion and processing are carried out. The resulting information is then used to decide some type of

control response for the system. This approach works well when the number of sensors is relatively small and the data packets being transferred are not large. However, when the number of sensors increases to hundreds or thousands, this centralized approach does not scale. The time frame in which data fusion/processing can take place will tax central processing units and may produce transient delays for desired control responses. This can quickly lead to degradation in system performance. Being able to effectively handle this flood of data to produce meaningful results will require new data distribution methods for sensor networks that are scalable, robust, and more efficient in resource consumption.

One way to distribute the work load is through the use of localized algorithms. A localized algorithm is a unique set of simple rules where coordination is achieved through local interactions of the individual agents. These interactions of the system's components do not rely on external control and are self-organizing. In this case, a sensor node can be thought of as an autonomous agent with some level of decision-making potential for the system. There are two distinct kinds of interactions for these types of systems: direct transfer of information between individuals and indirect information exchanges through the environment. The latter type of communication is termed stigmergy and was originally used to describe the natural behavior in coordinated swarm systems, e.g., societies of ants, bees, wasps, and termites. Stigmergy is a very interesting bio-inspired concept that might prove to be a useful control model for sensor swarms; however, constructing a rule-set that will produce consistent global behavior can be very challenging.

Localized algorithms are hard to design because of the inverse problem of producing a desired global behavior using agents that only have local knowledge. In addition, different choices of algorithm parameters can lead to unexpectedly varied results for the system. This

task becomes particularly complex with the lack of suitable test beds to run trials prior to field implementation. In these situations, a virtual framework can help facilitate the development of conceptual algorithms. Simulation provides an artificial environment that can be used to test, evaluate, and visualize algorithms in a cheap and efficient manner compared to traditional methods. In this way, many parameters can quickly be examined to determine the value of a prospective control algorithm for a given application.

At this time, there are no simulated sensor frameworks that can be easily adapted into custom applications. There are, however, many emerging robot software applications that enable virtual experimentation in robot and sensor systems. Actual sensor implementations in these frameworks are limited and usually only useful in robotic type applications. Nevertheless, benefit can be gained from examining the structure of these software APIs and how they incorporate sensor devices for parallel use in our simulation framework.

Combining both sensor nodes for gathering information and actuator nodes for manipulating the environment, multi-robot systems are a good surrogate for large sensor networks. A multi-robot system is fundamentally a sensor swarm with the added capability of actuation. Of particular interest to this project are robot applications that make use of distributed and autonomous control schemes.

To gain a better understanding of self-managing systems, this paper will focus on techniques that use autonomous construction agents to build two-dimensional structures. These techniques are based on an extended concept of stigmergy, in which environmental elements are used to coordinate swarm groups [Werfel and Nagpal, 2006]. Our interest in multi-sensor control for the deployment of potentially large numbers of sensors is the motivation for the design of this experimental framework. The goal is to create a generalized

virtual sensor framework that can easily be extended for use in a broad range of engineering applications. Information gained from this research will be used to create a set of software tools in VE-Suite, a virtual engineering software package, to assist in the development of complex sensor networks and controls.

CHAPTER 2. BACKGROUND

In engineering, having access to a broad range of information is vital in making decisions for complex systems. Sensor networks have been an important provider, collecting many different types of data over a broad range of applications. With current technological advances, the availability of small, low-cost sensors will revolutionize sensing parameters.

2.1 Sensor Networks

Scientific advances have always made a difference in warfare, and military applications have historically been a driving force behind R&D in sensor networks. Early sonar acoustic sensors, the Sound Surveillance System (SOSUS), were used to detect, classify, and track quiet nuclear submarines during the Cold War [Chong et al., 2003]. More sophisticated networks have been developed since the Cold War for submarine surveillance, and SOSUS is now used by NOAA to monitor seismic and animal activity in the ocean—an example of wartime technology transferring to peacetime objectives.

Another Cold War innovation that has undergone technological evolution is radar. Networks of air defense radars defended the continental U.S. and Canada during the Cold War. The modern day counterpart defends not only the perimeter, but the interior United States from acts of terrorism and war, as well as being able to deploy mobile ground and sea units to hot spots around the world. Today's radar capabilities include mobile ground radar units that employ forward-based sensors such as the Forward Based X-Band Transportable (FBX-T) Radar. "The radar will have a direct interface with the BMDS (Ballistic Missile Defense System) command and control system. The radar will perform surveillance autonomously or as cued by other sensors, and it will acquire, track and discriminate threat

missiles and missile components, and pass this information to other BMDS tracking, discrimination, and fire control radars downstream" [FBX-T, 2008].

Non-military applications of sensor networks have evolved to include large-scale wireless networks with either mobile or immobile sensor nodes for habitat monitoring, environmental monitoring (volcanic and earthquake activity), vital sign monitoring, and agricultural monitoring (temperature and moisture variations within a field). Some examples of these networks include:

- **Leach's Storm Petrel** at Great Duck Island, Maine, [Romer and Mattern, 2008]

  Easily disturbed by humans, sensor nodes enable biologists to monitor:

  - Usage patterns of nesting burrows
  - Changes in environmental conditions inside and outside of burrows during breeding season
  - Variations of breeding sites and parameters of preferred breeding sites

  Sensor nodes are installed inside burrows and on the surface to measure humidity, pressure, temperature, ambient light level, and presence of birds

- **ZebraNet**, Mpala Research Center, Kenya, [Romer and Mattern, 2008]

  Monitored behaviors of wild horses, zebras, and lions include:

  - Grazing patterns
  - Activity patterns
  - Interactions within a species and among different species
  - Impact of human development on the species

  Animals are equipped with sensor nodes to estimate position and speed of movement. Recorded data is collected by car or plane as it passes the animals.

- **Glacier Monitoring**, Briksdalsbreen, Norway, [Romer and Mattern, 2008]

  Sensor nodes are installed in drill holes at varying depths in the glacier ice and in the till under the glacier to measure pressure and temperature. A tilt sensor measures the orientation of the node. Sensor nodes communicate with a base station on top of the glacier. Radio communication through ice and water is a major problem.

- **Wind Farm off the Coast of England**, [Romer and Mattern, 2008]

  Sensor nodes measure pressure, temperature, conductivity, current, and turbidity to monitor the influence of the wind farm on the structure of the ocean bed. Sensor nodes are deployed from a ship that secures their positions to the ocean bed via an anchor. A cable connects each sensor node to a buoy on the ocean surface that contains radio and GPS equipment to circumvent underwater radio communication difficulties.

- **Argo Ocean Monitoring**, [Argo, 2008]

  Argo is an international collaboration to collect data year round from the ice-free global ocean through an array of profiling floats. The sustained observations aid in understanding and predicting climate models. Battery-powered autonomous floats drift most of the time at 1000 meters (to give a uniform velocity field). At 10-day intervals, the floats descend to around 2000 meters and then rise to the surface (an approximate 6-hour trip), taking about 200 measurements for temperature and salinity as they rise. Once they surface, satellites determine the position of the floats and receive the data transmitted by the floats. That information is shared with other users within 24 to 48 hours. The float then sinks back to its parking depth to repeat the

cycle again. The need for Argo observations is an ongoing one, but financial constraints are a concern.

Applications for networked sensors are as varied as the imagination and technology will allow. Improvements in sensor technology and wireless communications have made it cost effective to deploy large numbers of sensors. Once standardization of common software platforms is addressed, extensive development in massive networks can be expected to follow—similar to the Internet boom with its standardized protocol (TCP/IP) for application implementation [Zhao and Guibas, 2004].

Areas that will benefit from these technological advances include wireless sensor networks that can be used to monitor the condition of equipment to warn of failure or need of maintenance [Zhao and Guibas, 2004] and future NASA missions using highly autonomous spacecraft that "will enable radically different missions with significant onboard decision-making leading to novel science opportunities" with reduced risks and operations cost [Sherwood et al., 2006]. Sensor nodes will be built into walls, roads, and bridges remaining dormant until activated by a timer event, vibration event, or tilt event. Data can be analyzed after the event (i.e. after a building collapses) to build stronger earthquake-proof structures. [Haenselmann, 2006]

2.2 Management and Control

Adding additional sensors to a system does not necessarily lead to improvements in sensing performance. There must be an appropriate control scheme that coordinates information flow to produce meaningful results. Large sensor networks create considerable challenges for the management and control of data processing and communication. There are three main

classifications of management and control schemes applicable to sensor networks: *centralized*, *distributed*, and *self-organized* [Dressler, 2008]. The following sections give a brief description of these categories and their potential benefits and downfalls.

2.2.1 Centralized Systems

Systems that exhibit a well-defined central control process are said to be centralized [Dressler, 2008]. An example of centralized control in sensor networks might consist of a set of loosely-coupled sensors that continuously collect and transmit data to a central controller. Figure 1 depicts the basic principles of a centralized control system. Individual systems, S_i, report information to a central control process, C, which then performs the necessary fusion of data to compute an outcome for the application.

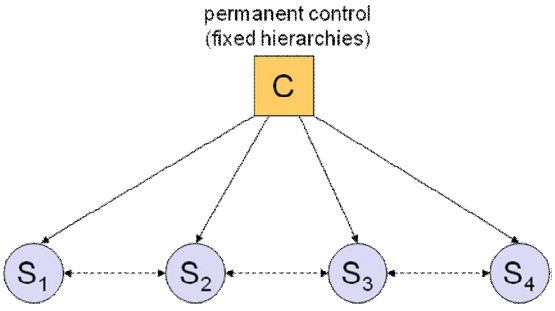

Figure 1: Centralized control principle (C depicts the control process and S_i the controlled systems, respectively) [Dressler, 2008, p. 11].

The control process is hardwired with all the properties of its subsystems, i.e. it knows about each of its subsystems and their respective capabilities. This trait leads to consistent and reliable behavior for centralized systems, but also makes it difficult to change the design of the system if certain parameters need to be reconfigured. Also, a centralized control process creates a single point of failure, which has the potential to incapacitate the

entire system. The centralized approach has two major weaknesses in relation to control and management of large sensor networks: scalability and transparency [Dressler, 2008]. Scalability refers to the ability of a system to add an increasing amount of sensors while maintaining a certain level of performance. Transparency refers to the ability of the system to adapt to changing resource availability. Computational limits for a central control process can quickly be reached when large amounts of sensor data become available, and its lack of flexibility does not permit for transparency in the system. Despite these weaknesses, a centralized system implementation is often the most cost-effective and time-efficient method when dealing with small numbers of sensors.

2.2.2 Distributed Systems

Distributed systems consist of a collection of independent subsystems that appears to an application as a single coherent system [Dressler, 2008, p. 10-11]. In this way, differences between each subsystem and the way they interact are effectively hidden from the user. This allows an application to interact with subsystems in a consistent and uniform way. To create this uniform interface between the heterogeneous subsystems, a middleware is typically utilized. Middleware is software that is placed between the application and the operating systems of each node. The middleware provides a common interface for components of the system to access information in the network. Middleware can be considered to be the glue that binds a network and its subsystems together. Figure 2 depicts the basic principles of a distributed control system. A specific control process, C, is dynamically allocated to one of the subsystems, S_i. This process maintains control until the resource request for the distributed system is complete.

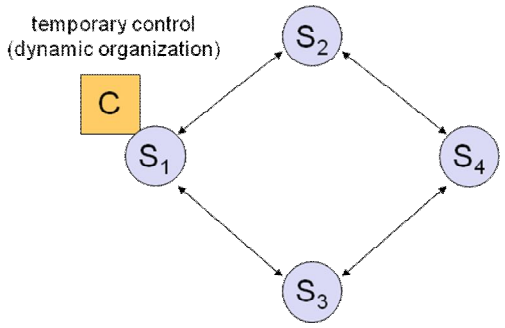

Figure 2: Distributed control principle (C depicts the dynamically placed control process and S_i the controlled systems, respectively) [Dressler, 2008, p. 13].

Distributed control provides three distinct advantages over a centralized process: adaptability, robustness, and scalability [Dressler, 2008]. Using the interface that middleware provides, all available resources are easily accessible from the application as well as from every subsystem. This abstract layer creates flexibility in integrating new components, even those that were not available during the original design of the system. Robustness is achieved through the distributed nature of the system. If no response is received from a failed node, the control process will be reallocated to another node in the network and tasking will continue. Some level of scalability is possible using distributed networks, but the subsystems must be limited in their ability to access non-local information. This restriction is not easily implemented for a large number of subsystems.

Research for large sensor networks typically focuses on distributed solutions for coordination due to the inherent dispersion of the sensors. Previous work includes distributed wireless ad hoc networks for tracking mobile targets [Chong et al., 2003], mobile agents as means to integrate data in distributed sensor networks [Qi et al., 1999], and various forms of distributed habitat monitoring [Romer and Mattern, 2008], [Mainwaring et al., 2002].

2.2.3 Self-Organizing Systems

Self-organization is a process in which structure and functionality emerge at the global level of a system due to the interactions among the lower-levels of a system [Dressler, 2008]. Each component in a self-organizing system has its own control process as shown in Figure 3. It is important to note that self-organization occurs solely based on the low-level interactions of the system and not by any external or central control process. The control scheme for this system can be compared to that of a centralized paradigm in that each component operates as its own system. The nodes of the system are completely self-dependent.

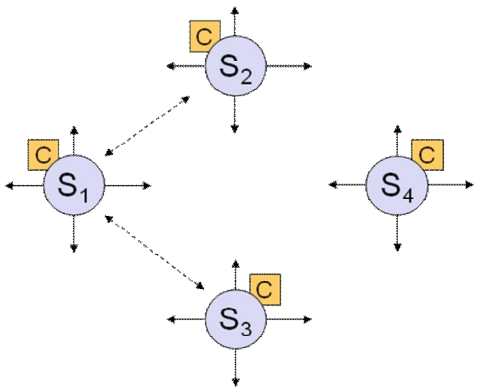

Figure 3: Management and control in self-organizing systems (C depicts the local control processes and S_i the controlled systems, respectively) [Dressler, 2008, p. 15].

Just as distributed systems provide advantages over a centralized process with respect to adaptability, robustness, and scalability, self-organized systems provide yet another level of performance for these properties. An intrinsic feature of self-organizing systems is the ability to adapt to changing conditions in the environment [Dressler, 2008]. Being able to make modifications in response to local environmental changes allows for regulation of the system's behavior. Self-organizing systems can also be extremely robust. Because each

component has its own control process, the failure of any one component in the system will not affect the behavior or performance of other components in the system. Scalability occurs as a result of components acting autonomously. While there are many advantages to be had from using self-organizing concepts, one downside is the difficulty in designing a behavioral rule-set that will produce a desired global behavior from components that only have local knowledge. A common approach to designing these rule-sets is to look at the self-organizing behaviors in nature.

Many principles of self-organization have been observed through the study of natural systems. These observations have inspired research for large sensor networks that mimic biological behavior, e.g., use of evolutionary algorithms [Das et al., 2006], artificial immune systems [Kim et al., 2001], and swarm intelligence [Di Caro et al., 2005]. Of particular interest to this research is the ability of swarms of social insects to behave collectively without any outside control. It is hoped that concepts derived from swarm intelligence will have potential use in coordinating information flow in large sensor networks.

2.2.3.1 Swarm Intelligence

Swarms of social insects can build exceptionally large and complex structures without a centralized control system and with seemingly limited perception. These swarms exhibit behavior on many levels above those of their constituent parts. For example, some species of termites can build nests 10^4 to 10^5 times larger than the size of an individual termite. (see Figure 4) [Bonabeau et al., 1988]. These nests display a high degree of regularity and contain complex regulatory systems, e.g., spiral cooling vents, nurseries, and fungus gardens. The

complexity of these types of systems is truly amazing and the underlying mechanisms are still not fully understood.

Figure 4: [Photo of Giant Termite Mounds in Litchfield National Park, 2008]

2.2.3.2 Stigmergy

Studies have shown that many of these structures are the result of a self-organizational process in which collective activities are coordinated through the environment [Theraulaz et al., 1999]. This principle was termed *stigmergy* by Pierre-Paul Grassé (1959) in an effort to explain his observations on the reconstruction of termite nests. Grassé observed that a particular configuration of the termite's nest would stimulate a response in workers to modify the construction. These modifications in turn would stimulate even further progressive responses that eventually led to the completion of the structure. The overall concept of stigmergy was depicted by Grassé's introduction to the term: "The worker does not direct his work, but is guided by it." [Holland and Melhuish, 1999, p. 174] Individuals react to

environmental variations that were produced by the previous actions of others. Information is stored implicitly within the environment and not directly transferred between agents.

The concept of stigmergy has crossed over into other domains of research including swarm intelligence, robotics, and network communications. This has led to a number of interpretations from its original context. Collectively, however, it can be viewed as a general explanation of how simple systems can produce a wide range of apparently highly organized and coordinated behaviors with behavioral outcomes, by exploiting the influence of the environment [Holland and Melhuish, 1999, p. 174]. Current research in the field of swarm robotics has shown the use of an augmented conception of stigmergy, *extended stigmergy*, which can improve system performance by increasing the capabilities of the environment [Werfel and Nagpal, 2006]. Their work focuses on creating autonomous algorithms that enable multi-robot systems to simultaneously build 2-D structures. This construction setting makes use of extended stigmergy by giving building blocks different abilities in which they can store information, perform computations, and communicate with attached neighbors. In this way, robots communicate implicitly by manipulating their surroundings. They extract information from a progressing structure's current configuration and use that information to decide upon new building actions. Extended stigmergy could be useful in coordinating multi-sensor systems in a variety of engineering applications and is explored further in this research by extending the framework developed by Werfel and Nagpal, 2006.

2.3 Virtual Framework

Through virtual simulation, many design parameters can be explored computationally without the cost and effort of building a physical model. Virtual reality presents three-dimensional, complex data in an intuitive and interactive manner that can help engineers understand the information better and make effective decisions. In the case of large sensor network self-management, a framework needs to be built that can help engineers understand what the value of additional sensors for a system will be. This application needs to support massive numbers of sensors so that engineers may explore and understand at what point are there too many sensors; how does the cost of the components impact the total system; how much information is enough to actuate the wanted results; what type of decisions need to be autonomous based on preset parameters; which algorithms are applicable to a situation; and any other questions that may arise during the virtual testing.

This research focuses on creating a virtual framework that will be helpful in gaining mathematical understandings for algorithms that support control and management of large sensor networks. This information can be used to identify which approaches are most likely to succeed when applied to the development of real world sensor applications. Algorithms that are proven to be applicable will be developed into reusable software tools that can be implemented in real sensing applications. Creating this open-source library will enable collaborative research to test, improve, and extend these tools. This investigative framework will be built within VE-Suite, an open-source virtual engineering software package being developed under the guidance of Professor Mark Bryden at Iowa State University. VE-Suite was chosen for its ability to create realistic environments and simple interfaces that can drastically reduce the turnaround time from a conceptual stage to functional system.

2.3.2 VE-Suite

VE-Suite is being developed as a multi-purpose virtual engineering design and analysis tool. The goal of virtual engineering is to create an environment that enables engineers to design, analyze, and modify their ideas. Toolkits that are successful in their design and implementation will be available in open source libraries for use by other researchers to further develop and test. An important aspect of VE-Suite is its ability to integrate many engineering components that may be necessary for an effective analysis and design of a system. This flexibility allows an engineer to couple together a number of simulation and modeling packages they may wish to use in the design process. This is accomplished by using an open communication interface called VE-Open. VE-Open combines CORBA (Common Object Request Broker Architecture) and XML implementations to communicate data in a self-describing data structure [Xiao et al., 2005]. This allows each component in VE-Suite's framework to query and access resources for the entire system in a consistent and uniform way. Differences between each component and the way they interact are effectively hidden from the user/application.

In order to take full advantage of the wide range of hardware resources a system implementation might include, the design platform of VE-Suite is divided into two parts: servers and clients. This architecture allows each server to run on a number of computational platforms, such as a Linux cluster, to promote task parallelism within the framework. The core components of VE-Suite consist of 3 servers: VE-Xplorer (the graphical engine), VE-Conductor (the user interface), and VE-CE (the computational engine) as shown in Figure 5. These core components interact with each other through the communication interface VE-Open provides.

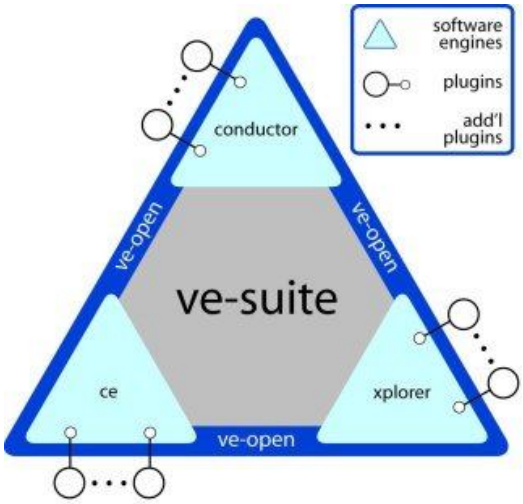

Figure 5: VE-Suite architecture

VE-Conductor is the graphical user interface (GUI) for VE-Suite. VE-Conductor provides a way for the user to interact with various parameters of the system to input and retrieve information for the system. *VE-Xplorer* is the graphics engine in VE-Suite's framework. Its main task is to provide a visual experience for the user that conveys information in a 3-D virtual environment. This information can be anything that is useful to the design process, e.g., numerical models, CAD geometry, CFD data, and experimental data. *VE-CE* is the computational engine for VE-Suite. VE-CE handles the scheduling and synchronization of data sources that are being used for a particular application [McCorkle et al., 2007]. VE-CE is responsible for issuing commands from the application/user to a specific computational unit that processes its command and relays the information to the appropriate module.

Other servers and clients that adhere to VE-Open's communication specifications can also be "plugged" into VE-Suite's framework to provide or access resources to/from the system. This allows each component in the software framework to be aware of other models for communication and retrieval of information [Xiao et al., 2005].

2.3.3 Software for the Support of Multi-Robot Systems

Many software packages currently exist to facilitate experimentation in multi-robot systems. Some of the beneficial traits these packages may comprise are: commonly defined interfaces for software components that are frequently reused, a set of low-level drivers for robot hardware, a middleware library that allows local and networked communication efficiently between components, and a simulation environment to substitute when hardware is not necessary or available [Michael et al., 2008]. The following are some examples of open-source distributed robot software frameworks. These libraries provide a similar framework to what is described in this research and will be investigated for their potential to support a large number of sensors in virtual simulation.

2.3.3.1 Player/Stage/Gazebo

Originally created at the University of Southern California for ActivMedia Pioneer 2 robots, open-source Player/Stage/Gazebo has evolved to support numerous robotic platforms, hardware devices, software systems, and algorithms [Gerkey et al., 2003]. Player is a widely-used server that allows for the control of networked robots. Stage simulates a two-dimensional environment for multiple robots to engage, and Gazebo is a three-dimensional multi-robot simulator for outdoor environments [The Player Project, 2008].

With Player running on the robot, communication between Player and the client program takes place via a transmission control protocol (TCP) socket [Gerkey et al., 2001], [Gerkey et al., 2003]. Player allows data to be exchanged among drivers and control programs that are running on different machines and is designed to support any number of clients. A recent innovation is the development of abstract driver allowing users to create their own drivers instead of being confined to hardware-specific drivers [Gerkey et al., 2003]. Stage is designed to simulate a large population of robots, objects, and sensors—including sonar, scanning laser rangefinders, vision (color blob detection), odometry, and a differential steer robot base—in 2-D, but sacrifices fidelity. Gazebo is designed to simulate a small population in 3-D with high fidelity. Gazebo uses ODE (Open Dynamic Engine - a rigid body dynamics simulator) to compute interaction with objects [Koenig and Howard, 2004].

Player/Stage/Gazebo is a powerful robotics software package, but it relies on several complex components and has a steep learning curve. It also lacks the robustness, extensibility, and network capability needed for the test platform being built here.

2.3.3.2 CARMEN

The Carnegie Mellon Robot Navigation Toolkit (CARMEN) is an open-source collection of modular software for mobile robot control that communicates via a central server to navigate a robot or map new environments, but has been also used in applications such as health care robotics, person–tracking, and mine exploration [Carmen Robot Navigation Toolkit, 2008], [Montemerlo et al., 2003]. Communications between CARMEN executables use InterProcess Communication (IPC), which is a separate software package but is distributed with

CARMEN. Carmen was designed to encompass extensibility and robustness but its downfall is that it is very complex and is limited to mostly indoor robots/vehicles [Teller, 2005].

2.3.2.3 Orocos

Frustration with the available commercial software for advanced robotic research was the motivation behind the idea for a free software project named Orocos, Open Robot Control Software. The sponsored project started in 2001 and lasted for two years focusing on industrial robotic real-time applications. Orocos was primarily developed by Katholieke Universiteit Leuven in Belgium, the Laboratory for Analysis and Architecture of Systems (LAAS) in France, and Kungliga Tekniska Hogskolan (KTH) in Sweden, with contributions continuing from various countries. Orocos is a modular framework that utilizes CORBA as a middleware, libraries for kinematics/dynamics, and Bayesian filtering. Applications include automation and robotics with components for sensor data processing, industrial machining, kinematics, control algorithms, and hardware access. The component-based robotic system spinoff was renamed Orca [The Orocos Project, 2008].

2.3.3.4 Miro

MIRO (Middleware for Robots) was originally developed at the University of Ulm in Germany in 1999. MIRO is a CORBA-based framework for programming mobile robots. The CORBA interface allows for data exchange between modules written in different languages. MIRO also provides a Player interface so that MIRO software can be tested in Stage and Gazebo simulators. Like many multi-robot software packages the MIRO/ACE (a distributed programming library)/TAO toolchain has a fairly steep learning curve, but reduces software development time and expense by providing robot-specific data structures,

functions, communications protocols and synchronization mechanisms [Kuo and MacDonald, 2004], [Broten et al., 2006], [Kruger et al., 2006].

2.3.3.5 MARIE

No mobile robotics standards exist. Linking applications having different communication protocols and communication mechanisms requires time, effort, and knowledge to assure that functionalities are not deteriorated or modified. MARIE (Mobile and Autonomous Robotics Integration Environment) is an open-source middleware framework that reuses software, APIs (Application Programming Interface), middlewares, and frameworks (such as Player and CARMEN) to avoid code duplication in robotic systems components. Developers are able to more quickly design prototypes when they can borrow ready-made components and integrate them for their specific requirements, [MARIE, 2008]. Unfortunately, MARIE's scope is limited to robotics and is suffering from decreasing development activity—down to a single, active developer according to one website [Cote et al., 2004], [MARIE, 2008].

CHAPTER 3. FRAMEWORK

This research project is focused on developing a simple framework to manage coordination of autonomous construction agents. Central to this research framework is enabling the inclusion of stigmergy in which a set of low-level rules that controls the construction agents behavior are implemented through various environmental signals. In this research the construction agent problem serves as a surrogate problem in understanding information control schemes with virtual sensors. The goal is that through this framework, software tools will be developed for VE-Suite that will aid in the design of future sensor network applications.

3.1 Design Implementation

The basic framework being developed here for the construction problem is, to a great extent, the same as the original design (see Figure 6) developed by Werfel and Nagpal, 2006. Building blocks are unit cubes that can be attached to each other on sides corresponding to the desired two-dimensional structure. One building block is initially positioned to indicate where construction should begin. Mobile agents locate blocks, take them to the structure, and follow its perimeter until they find a valid attachment site. To prevent construction interference, unattached blocks are randomly placed a “safe” distance away from the structure.

In this construction setting, the building blocks are the environmental elements that are given increased capabilities, i.e. extended stigmergy. They have the ability to store information, perform computations, and communicate with physically attached neighbors.

Later we will discuss the algorithm, *Communicating Block*, used to ensure constraints on block attachment.

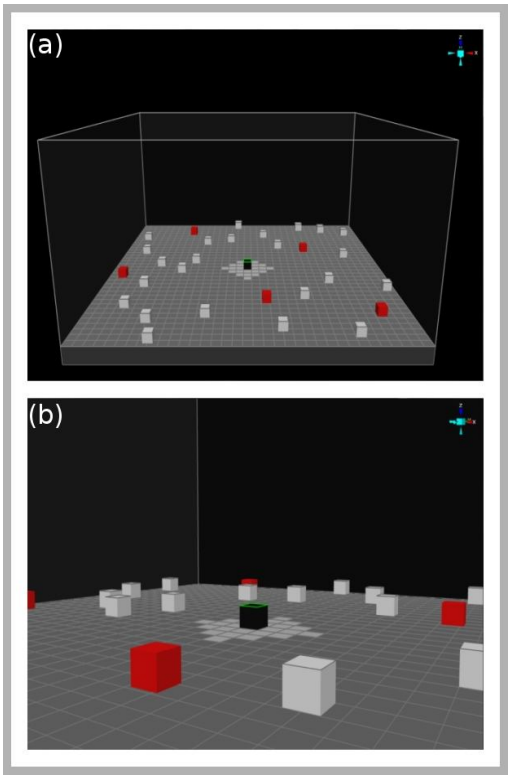

Figure 6: Construction problem framework implemented using VE-Suite. The virtual construction world consists of mobile agents (red), building blocks (white), and a block-sized marker to designate the starting point for construction (black) [Werfel and Nagpal, 2006].

3.1.1 Virtual Environment

An important element of creating a realistic virtual environment is providing an effective visual interface for the user. The construction problem framework implements this interface by using VE-Suite [VE-Suite, 2008]. VE-Suite can create a fully immersive virtual environment in which decision-making processes can take place or simply provide a two-dimensional virtual workspace. Through this interactive graphical interface, visual

recognition of data and information flow gives the user an enhanced connection and understanding of the environment. For the study at hand, this may allow innovative adjustments to the informational control schemas. VE-Suite provides “plugins” to each of its core components so that custom applications can be designed on top of VE-Suite’s default capability. The framework developed in this research makes use of VE-Suite’s graphical plugin interface to VE-Xplorer. The graphical plugin provides access to VE-Xplorer’s toolset for developing a virtual environment.

3.1.2 Physics Engine

The open-source physics package Bullet [Bullet Physics Library, 2008], a fully capable collision detection and rigid body dynamics library, is integrated into VE-Suite for the purposes of this framework. Through this interface, an initial step is taken towards reproducing the dynamic environment an agent may encounter, including its physical attributes such as mass, velocity, and friction. In the current physics implementation, elements are constrained in their “degrees of freedom” to ensure alignment with the grid structure and still retain continuous valued positions. Other forces are being simulated for the agents as they move over the two-dimensional ground plane, including those created by obstacle avoidance algorithms and gravity.

3.1.3 The Construction World

Certain assumptions were made in the original framework developed by Werfel et al., 2006, pertaining to the capabilities of the agents. It was assumed the agents could perform the following tasks:

1) They are able to move in any direction and avoid collisions even while holding a block.
2) They are able to follow beacons to get to the building site and to block locations.
3) They are able to perimeter follow the building site and perceive turning a corner.
4) They are able to pick up blocks from caches and attach them to the structure.

Many of these assumptions are addressed for the current framework to create a realistic simulation environment. While it is not completely necessary to simulate all aspects of the construction problem, adding these details to the system would make the environment more interactive and at the same time enable more diverse informational configurations in the framework. This endeavor initially incorporates the integration of collision detection and rigid body dynamics, virtual sensors, and simplistic obstacle avoidance algorithms for the agents. Following is a brief description of each general component involved in creating the complete simulation.

3.1.3.1 Models

A model is any object that represents a physical entity. There are three types of models in this framework representing the construction agents, the building blocks, and the construction environment. All models in this framework consist of a geometric shape and at least one rigid body. A rigid body is the physics representation of the model and is used in calculating physical forces by the physics engine. Each rigid body has associated physics properties including mass, friction, and restitution while geometric bodies contain rendering properties such as material color, texture, and opacity.

The boundaries of the construction world consist of a ground plane that the agents and building blocks rest upon and surrounding walls that are used to confine the dimensions of the construction environment. Gravity is the working force that keeps the agents and building blocks held against the floor, while an obstacle avoidance algorithm is implemented by agents to avoid hitting the walls. This ensures that the components of the system will occupy a space that is relevant to the construction task at hand. The ground plane and walls represent static models because they are stationary in the environment. They are not affected physically by any other models in the scene. Therefore, forces are not calculated for these bodies.

Building blocks are simple box-shaped geometries. The blocks represent a dynamic model because they can be affected by physical forces in the environment. These may include gravity, collision with other objects in the environment, or a user induced force through the virtual interface that VE-Suite provides.

Agents are also simple box-shaped geometries, but, unlike the building blocks, agents contain virtual sensors to gather information in the construction world. They use this information to decide upon their behavior for the construction task. The agents are dynamically represented and have the added ability to move across the ground plane. This movement is simulated in the virtual environment by applying small forces through the physics API.

3.1.3.2 Sensors

Important environmental information needed to direct mobile agents can be obtained from sensor devices such as cameras, infrared and ultrasonic sensors. For this application, the

agents require a way to find the blocks and the construction site and simultaneously avoid unnecessary obstacle collisions. To enable this behavior, different types of virtual sensors are created to aid the agents in these tasks. To simplify their implementation, sensors are not given any physical representation in the framework. They are an abstract class whose sole purpose is to collect data for the agents. This design keeps the interface between the agent and each of its sensors simple and allows for reuse of code among the different types of sensors. Since sensors only collect data and do not represent a physical entity, they are not included in the physics simulation. The framework initially incorporates three different types of sensors—including ray proximity, sonar, and color detection.

The ray proximity sensor returns the distance from the closest contact point of an object along the ray's path. This sensor is meant to emulate the large number of range finders available for robotic experiments. This sensor is fairly easy to implement in code using a basic ray intersection test with the geometry in the environment.

The color detection sensor collects various color information from the scene. To permit the agents to sense this information, a virtual camera implementation is simulated in code to give the agent color feedback of its environment. To allow the agents to differentiate between blocks that are attached to the structure and blocks that are unattached, the blocks are colored different depending on their current state of attachment. In this way, the color detection sensor can be used to direct an agent towards building blocks and then back to the construction site. This may not be realistic behavior for an actual building block, but it serves its purpose for the testing of the framework.

The sonar sensor detects possible obstacles that the agents may encounter as they perform tasks. A real sonar sensor functions by emitting sounds and waiting for a return

echo. By keeping track of the time it takes for a signal to return, a distance can be calculated for each obstacle hit. To keep our agents relatively uncomplicated, a simplified concept of the sonar sensors is simulated in code by creating a circular pattern of ray intersection tests. If a ray contains multiple intersections, only the first intersection is used as would be seen in a sonar reading. These readings are then fed into various real-time obstacle avoidance algorithms that calculate a total repulsive force in a direction away from the highest concentration of obstacles.

3.1.3.3 Obstacle Avoidance

Navigation in autonomous systems implies that agents have no prior knowledge about potential obstacles in their path. In this framework, agents will need to successfully navigate other moving agents as well as blocks that are scattered throughout the environment. This will require a dynamic obstacle avoidance system that can detect moving and static objects and use this information to calculate a direction of motion that allows an agent to complete its tasks. Obstacle avoidance algorithms adapted from Borenstein and Koren, 1989, employ the idea of obstacles conceptually exerting forces onto the mobile agents as they approach. This repulsive force is then combined with a target force, representing an agent's goal position, to produce a resultant vector of desired travel for the agent.

There are two obstacle avoidance algorithms implemented in this application: *Virtual Force Field Method (VFFM)* and *Wall-Following Method (WFM)* [Borenstein and Koren, 1989]. Readings are continuously taken from the sonar sensors while an agent is moving. As the agent comes in range of an obstacle, its sonar sensors register a detection and record the distance information for the reading. In this way, a repulsive vector can be calculated with

respect to the agent. This information is then sent to the *VFFM* algorithm where it is merged with the target vector to produce a resultant vector. If the resultant vector points in a direction 90^o away from the target, the *WFM* algorithm is invoked to prevent the agent from getting caught in a local trap. This allows an agent to actively adjust its path to avoid collisions and does not require an agent to stop while it performs tasks. An unattended side-effect of these repulsive forces is the distributed coverage that naturally occurs as the agents operate. Being repulsed from each other keeps agents from occupying the same space and limits redundant sensor readings. A full mathematical description of the obstacle avoidance algorithms can be seen below.

Virtual Force Field Method (VFFM): As an agent approaches an obstacle, it is pushed away by a repulsive force proportional to its distance from the obstacle. The magnitude of the sum of all the repulsive forces from the environment gives a total repulsive force, $\vec{F}_r$, for the agent as shown in Equation 1 [Borenstein and Koren, 1989, p. 4]:

$$\vec{F}_r = \sum_{i,j}\left[\frac{F_{cr}}{d^2(i,j)}\left(\frac{x_i - x_0}{d(i,j)}\hat{x} + \frac{y_j - y_0}{d(i,j)}\hat{y}\right)\right] \tag{1}$$

where
$\vec{F}_{cr} =$ Force repelling constant
$d(i,j) =$ Distance between an obstacle detection and the agent
$x_0, y_0 =$ Agent's present coordinates
$x_i, y_j =$ Coordinates of an obstacle detection with respect to the agent

A constant-magnitude attracting force, $\vec{F}_t$, can be implemented if the agent has a target point and its coordinates are known with respect to the agent as shown in Equation 2 [Borenstein and Koren, 1989, p. 5]:

$$\vec{F}_t = \sum_{i,j}\left[F_{ct}\left(\frac{x_t - x_0}{d(t)}\hat{x} + \frac{y_t - y_0}{d(t)}\hat{y} \right) \right] \tag{2}$$

where
$\vec{F}_{ct} =$ Force attraction constant
$d(t) =$ Distance between the target and the agent
$x_0, y_0 =$ Agent's present coordinates
$x_t, y_t =$ Coordinates of the target with respect to the agent

The vectorial sum of all these forces produces a resultant force vector, $\vec{R}$, in Equation 3 [Borenstein and Koren, 1989, p. 5]:

$$\vec{R} = \vec{F}_t + \vec{F}_r \tag{3}$$

Wall-Following Method (WFM): A potential downfall of the *VFFM* algorithm is the possibility of an agent to get trapped as it performs tasks (see Figure 7a). To compensate, a different algorithm is employed when an agent detects that it is in a trapped state. This algorithm is known as the *wall-following method* (see Figure 7b).

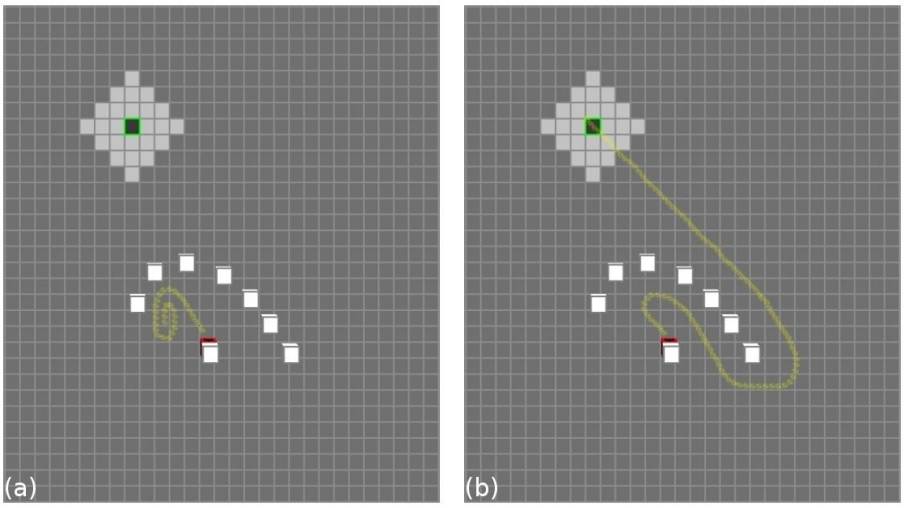

Figure 7: (a) The agent gets caught in a local trap. (b) As the agent finds itself in a direction 90**º** off target, it switches to the *WFM* algorithm. Once the agent's heading is back within 90**º**, the *VFFM* algorithm is reinstated.

An agent recognizes it's in a trapped state if the resultant force vector calculated by the *VFFM* algorithm takes the agent in a direction 90^o away from the target. Once this occurs, the agent switches to *WFM* mode. The *WFM* algorithm first calculates the sum of all repulsive forces, $\vec{F}_r$, in the same manner as the *VFFM* algorithm. Next, the *WFM* algorithm rotates an angle $90^o < \alpha < 180^o$ away from the total repulsive vector, and a virtual attractive force, $\vec{F}_t$, is projected in this new direction. The resultant force, $\vec{R}$, will now point and eventually converge in a direction parallel to the wall boundary as shown below in Figure 8. Once the agent escapes the trap and is headed back in a direction within 90^o of the target, *VFFM* mode is resumed.

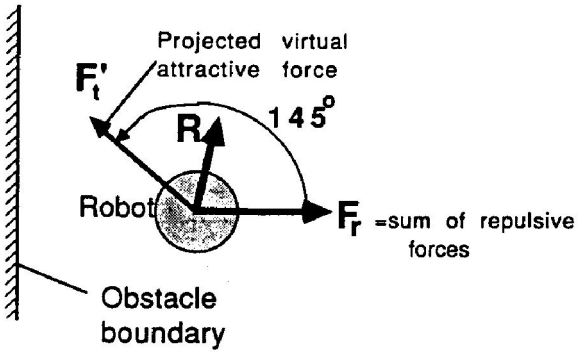

Figure 8: The *wall-following method* used by the agents to avoid local traps [Borenstein and Koren 1989, p. 12].

3.1.3.4 Behaviors

Agents are now able to gather information from their environment through use of virtual sensors. In addition to this, some behaviors need defined to encompass tasks an agent can perform. These behaviors are based on the simple rules determined by the governing control algorithm. Collectively these components will form the basis of a behavior control scheme for our agents. This scheme is adapted from a research application pertaining to collective construction in autonomous mobile robots [Wawerla et al., 2002].

The information that each virtual sensor gathers, pertains to a primary response for the agents. Also, each sensor has the ability to influence other behaviors in a secondary role. Later, a flowchart for the control scheme will be discussed in more detail, but for now a brief description of the sensor/behavior relationship can be seen below in Figure 9.

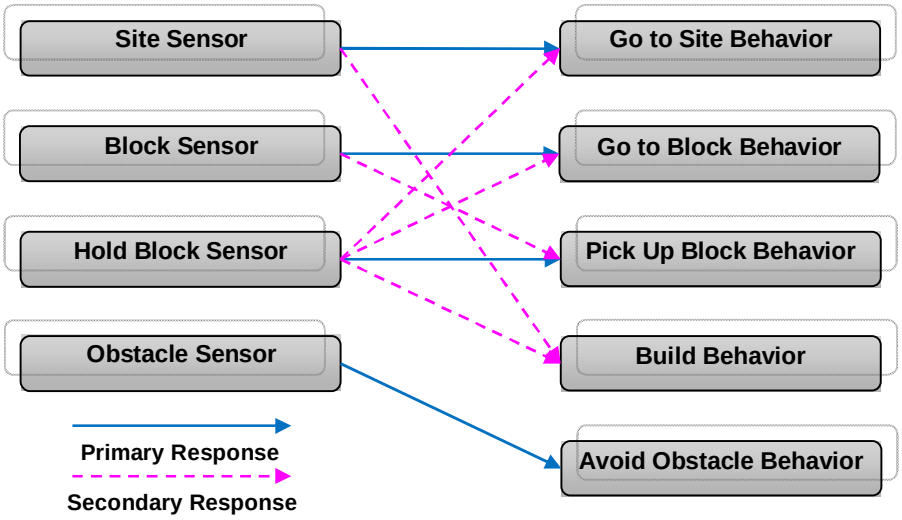

Figure 9: Simple model of the behavior-based control scheme for the agents [Wawerla et al. 2002].

Sensors

- *Site Sensor* - returns distance and direction information when the site is in view
- *Block Sensor* - returns distance and direction information for blocks in view
- *Hold Block Sensor* - tells the agent if it is holding a block
- *Obstacle Sensor* - returns distance and direction information from detected obstacles

Behaviors

- *Go to Site Behavior* - directs an agent towards the construction site
- *Go to Block Behavior* - directs an agent towards a block
- *Pick Up Block Behavior* - directs an agent to pick up a block
- *Build Behavior* - causes the agent to perimeter follow the construction site and connect blocks at valid attachment sites
- *Ovoid Obstacle Behavior* - implements obstacle avoidance algorithms to prevent obstacle collision with blocks and other agents

CHAPTER 4. SENSOR TEST CASE

To test the framework, a series of experiments will be performed to demonstrate the successful construction of a structure in virtual simulation. A stigmergic algorithm will be implemented as the control scheme to organize the behavior of the agents. The variables for this test case are: a) the number of agents that participate in the construction task and b) the distance in which agents can "see" the building blocks. To enable performance analysis, the time it takes to successfully complete the structure will be recorded.

4.1 Algorithms

In Werfel and Nagpal, 2006, three algorithms that make use of extended stigmergy are described and analyzed. For our initial work, we will focus on the one termed *Communicating Blocks*.

4.1.1 Communicating Blocks Algorithm

This algorithm makes use of a decentralized controller by embedding a processor in each building block. Blocks maintain a copy of the desired shape (an occupancy matrix), information about their location in the shared coordinate system, and side state based on bordering attachments. Once a block is attached to the structure, it can communicate with its immediate neighbors through the physical connection they share. This new block gets a copy of the desired shape and its location from its neighbors, and sets its side states according to theirs. The other blocks then update their side states based on this new attachment. This design allows the structure to enforce desired restrictions on block attachment (see Figure 10).

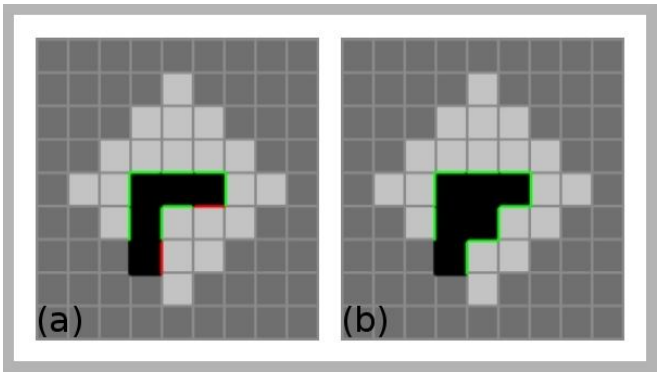

Figure 10: (a) Blocks form a data-line to communicate to agents where they can attach a new block (green) or sites that are unavailable for attachment (red). (b) A new attached block sets its side states based on its neighbors, and they update theirs accordingly [Werfel and Nagpal, 2006].

4.1.1.1 Separation Rule

To ensure the desired structure is built properly, a partial ordering on block attachment (the *separation rule*) is set forth for the *Communicating Blocks* algorithm. The *separation rule* states that separated blocks must never be attached in the same row (that is, with the same x- or y-coordinate) if all sites between them are ultimately meant to be occupied [Werfel and Nagpal, 2006, p. 22]. The need for this rule arises from the fact that we are dealing with rigid building materials (the blocks). For example, it would be difficult for an agent to fit a building block into a space that is exactly one block wide. The separation rule also prevents long narrow tunnels from forming in which it is impossible for two agents to pass each other while perimeter following. Undesirable situations like these can be seen in Figure 11.

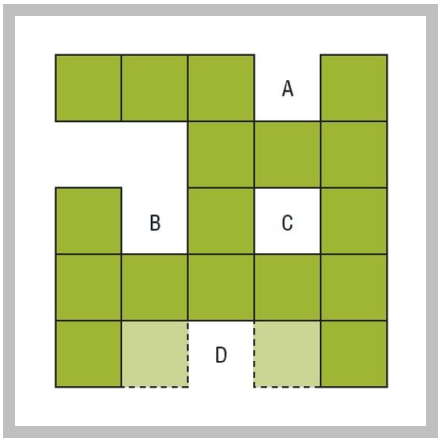

Figure 11: It is assumed impossible for an agent to insert a block into a tight opening like (A). To avoid this situation, separated blocks must never be placed in the same row if all sites between them are meant to be occupied (D). Following the *separation rule* prevents other situations like (B) and (C) [Werfel and Nagpal, 2006, p. 25].

It follows that if we wished to build a structure with the physical traits shown above, this would not be possible using the *separation rule*. Therefore, suitable structures using this approach must be free of intentional holes and concavities must be two blocks wide to allow perimeter following by the agents. These provisions prevent situations in which agents physically cannot reach sites meant to be occupied.

4.2 Experimental Design

All simulation experiments are performed in VE-Suite. An environment consists of a ground plane measuring 51ft. x 51ft. with surrounding walls that are the same dimension in height. The walls are detectable by the agents' ray proximity and sonar sensors as would be the case in a real environment. Agents and blocks are both unit cubes measuring 1ft. x 1ft. x 1ft. The size of these elements remains constant for the entire study at hand. The desired shape being built is stored as an occupancy matrix—a Boolean map—and can be seen below in Figure 12. The occupancy matrix can be any shape, but this particular shape is chosen for its symmetry with respect to the environment. In this way, the space around the structure is consistent, and the effects on an agent's obstacle avoidance system are minimized.

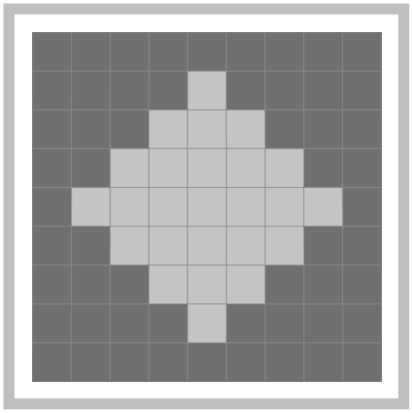

Figure 12: The occupancy matrix used in this test case.

Twenty-four spaces are to be occupied to complete the structure as can be seen in Figure 12. An initial block is placed at (0, 0), the center of the ground plane, as a marker to designate the starting point for construction. The start block is given the occupancy matrix, and its location and side states are updated accordingly. For each agent and block, the

physical properties mass, coefficient of friction, and coefficient of restitution are set to 1.0kg, 0.5, and 0.0 respectively. Since the ground plane and walls are static, only their friction is set at a value of 0.5. The 23 blocks left to complete the structure and the agents are randomly placed in the environment a “safe” distance away from the structure so as to not interfere with construction. Finally, the range is set for the agents’ sensors to limit what they perceive in the environment.

The agents have absolutely no prior knowledge about the environment they are placed in. Agents are identical and unperceptive to the status of the construction task in which they participate. They build structures by moving blocks, coordinating their actions solely through the progressing structure. This implicit communication eliminates the dependence upon other agents and enables a single agent to complete the construction task by itself demonstrating (reflecting) robustness of the system.

The flowchart, shown in Figure 13, represents a simplified behavior scheme for the agents. There are two sets of conditionals in this design. The first set of conditionals decides the construction tasks an agent should perform. While this occurs, the *obstacle sensor* continually monitors for obstacles in the path of the agent. If obstacles are detected, a repulsive force is calculated and combined with the target force produced in the first set of conditionals. If no target or obstacle is present for the algorithm, the agent will move in its current direction until either an obstacle or target causes it to change. A brief description of the thought process an agent goes through is discussed.

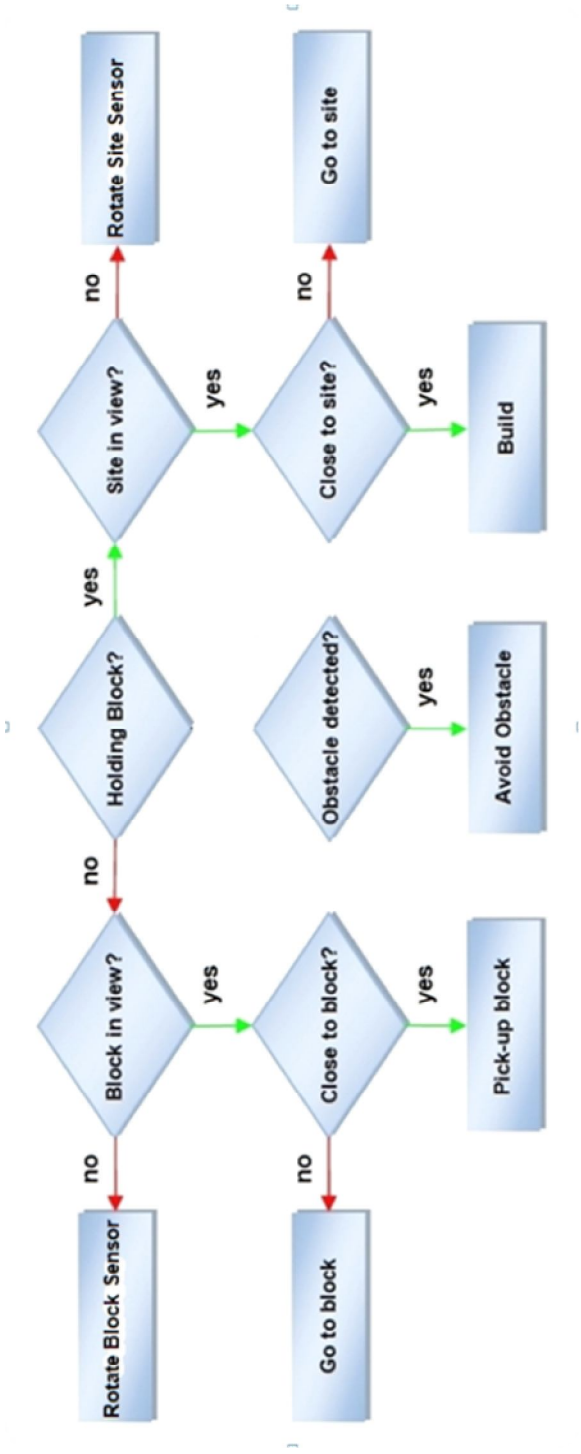

Figure 13: Simplified flowchart of the behavior-based control scheme for the agents.

As an agent is placed in its new environment, the first task it addresses is whether or not it is holding a building block. Given that the agent has not yet had time to find a block, the *hold block sensor* registers false. Based on this information, the agent knows it needs to locate a block for the construction task. It then queries its *block sensor* for information about the successful return of color data matching a construction block. If a block is not in site, the *block sensor* is swept continually as the agent moves until a successful detection is registered. The agent identifies the direction of the block with respect to its own coordinate system and moves in the target's direction. As the agent approaches the block, the *block sensor* updates distance information so the agent can make a proper proximity analysis. Once sufficiently close to the block, the agent will pick it up.

Now that the agent is holding a block, the *hold block sensor* registers true. This information then directs the agent to find the construction site. The *site sensor* is queried in the same fashion as the *block sensor* until the site is successfully located. Again, the agent identifies the direction of its target and moves towards the site. Once the *block sensor* informs the agent it is close to the construction site, the build behavior is initiated. The build behavior is inherently more complex than other behaviors and is a cooperative process between the agents and blocks. Both procedures are summarized in the pseudocode below in Figure 14:

```
A: Blocks

loop
  for sides S do
    if (robot asks to attach a block to S) then
      if (occupancy matrix specifies a block there) and
        (separation rule specifies a block there) then
        allow attachment
      else
        forbid attachment

B: Robots

while (structure not complete) do
  if (holding a block) and
     (obstacle is not detected) then
       ask adjacent structure blocks if block can be attached here
    if (structure block answers yes) then
      attach block here
    else
      follow perimeter counterclockwise
  else
    exit build mode
```

Figure 14: Pseudocode encapsulating the build behavior between the agents and the blocks [Werfel et al. 2006]

To illustrate Figure 14, once an agent enters build mode, it docks the construction site and queries blocks for valid attachment sites. If permission is not granted, the agent begins to perimeter follow in a counterclockwise manner. The agent will continue to perimeter follow the structure until either a block gives permission to attach or an obstacle is detected (such as another agent trying to dock). For the latter case, the agent will halt its build behavior and switch to obstacle avoidance behavior to safely avoid a collision. Once the collision is averted, the agent will try to locate the construction site again. As was stated before, when a

new block is attached to the structure, it sets its side states based on those of its neighbors, and they update theirs based on this new attachment. The *separation rule* is invoked while setting the side states to ensure the completion of the structure.

4.3 Simulations

Numerous two-dimensional structures are built using the *Communicating Blocks* algorithm to get a reasonable statistical base for each simulation. The number of agents and the range in which they can "see" building blocks are varied throughout the experiments to evaluate performance based on sensor capability and multi-agent tasking. The number of agents varies from 1 to 24 while the *block sensor* range varies from 10ft. to 70ft.

The maximum number of agents is limited to 24 to represent the total number of blocks in the environment. It is presumed that adding more agents will only hinder overall performance for the system since not all agents will participate in the construction task. Also, calculations are handled in series for the agents on a single CPU since the framework is lacking a parallel implementation. This quickly overloads most computers when simulations are run for more than 20 agents. Future research will focus on integrating parallel interfaces in which devices can be hooked into VE-Suite's framework.

The maximum *block sensor* range of 70ft. is chosen because it represents the longest distance a building block can be from an agent in a 51ft. x 51ft. environment. The *block sensor* range starts at 10ft. and is incremented by 2ft. for each set of simulations. This gives a total of 30 different test lengths for the *block sensors*.

For each combination of the number of agents and *block sensor* range, 40 simulations are taken. In all, this gives a combined total of 28,800 total simulations for the test case (see

Appendix. Statistical Results). The time it takes to complete the structure is measured by the number of rendered frames, which assumes that all agents perform a behavior in the same amount of time each frame. This effectively normalizes the computation speed on processors with differing capabilities to allow for multiple simulations to be run on a number of computers. A complete simulation run for one of these structures is shown below in Figure 15.

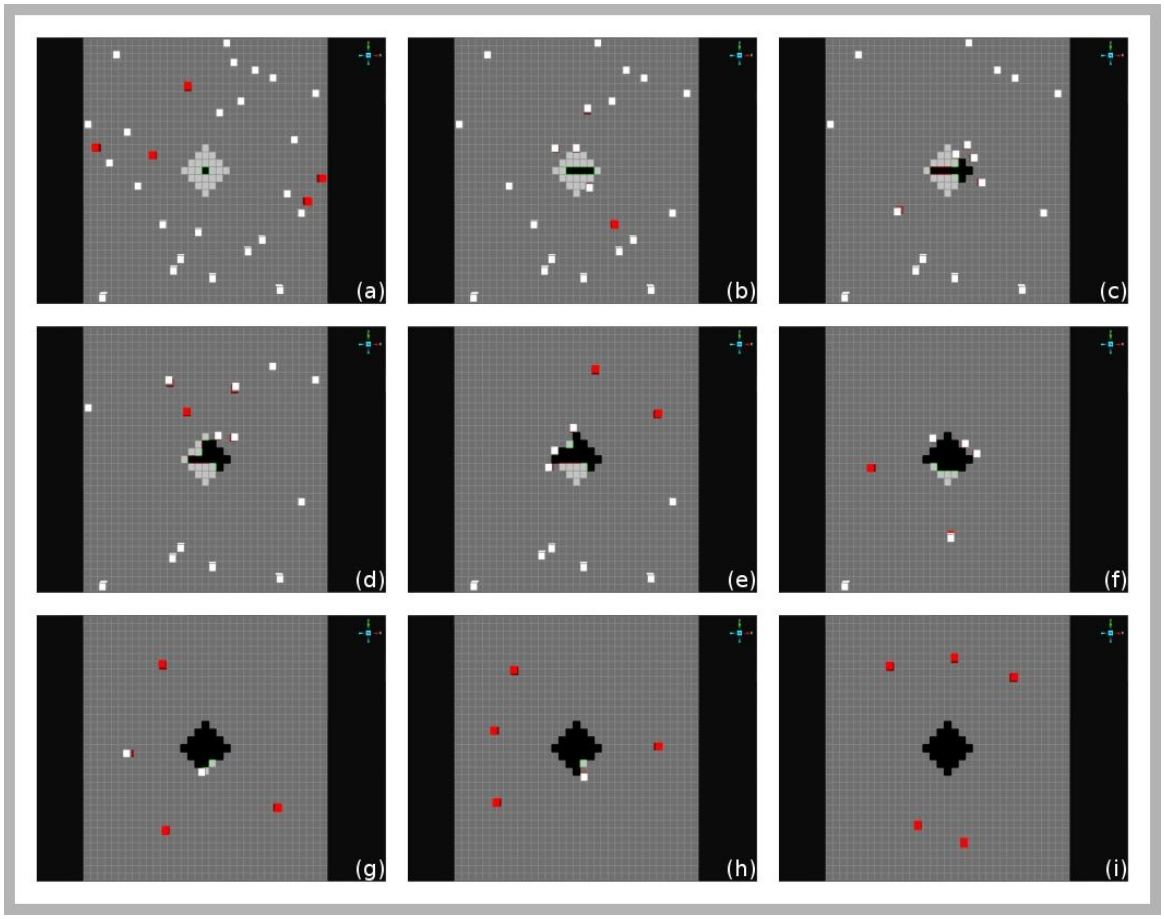

Figure 15: A predefined structure under construction using the *Communicating Blocks* algorithm. Lightly shaded regions show the desired shape of the structure to be built.

CHAPTER 5. RESULTS

The *Communicating Blocks* algorithm is analyzed for possible functional relationships to classify its performance for the construction task. This information may be useful in gaining mathematical understandings for particular classes of autonomous algorithms. These trends can also be used to identify which approaches are most likely to be successful when applied to a particular engineering application.

5.1 Algorithm Effectiveness

One way to validate the performance of a large number of swarm elements is to first examine the effectiveness of a single agent to complete the task. It is expected that the value of additional capability for the system will initially rise as more agents are added to the swarm. Eventually though, performance gains will diminish, as additional agents will no longer add significant contribution to the system. This all depends on the task at hand and the controlling mechanisms for the swarm.

This trend is shown in NASA research that enables bio-inspired sensor swarms to detect, isolate, and repair atmospheric pressure leaks in the International Space Station [Fronczek and Prasad, 2007]. These leaks occur when space debris collides with the space station. They rationalize if a leak is large, less swarm sensors are needed to detect the pressure drop associated with the turbulent flow of escaping air. In this case, a large number of sensors cannot add any additional value to the task of finding the leak. But if the leak is small, the dynamics of the air-flow will not easily be detected and will therefore require a larger number of swarm sensors to locate and isolate the fault. In this case, it is beneficial to deploy more sensors. Figure 16 illustrates the functional relationship for the time it takes N

number of swarm individuals to successfully locate a leak source of a certain size. The exponential decay in performance eventually leads to diminished returns for the sensor system. This trend coincides with results from the experiments for this test case.

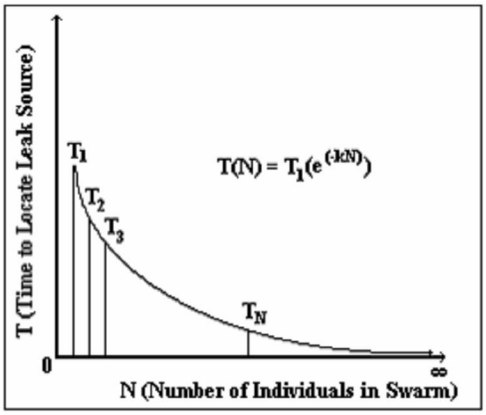

Figure 16: Time it takes N number of swarm sensors to locate a leak source [Fronczek and Prasad 2007, p.1].

Referring to Figure 16 a functional relationship of the form shown in Equation 4 can be realized:

$$T(N) = A + T_1 e^{-KN} \tag{4}$$

where
$T_1 =$ The time it takes for one sensor to perform the task of leak detection
$N =$ The number of swarm sensors
$K =$ A swarm effectiveness constant based on the parameters within the system
$A =$ The asymptotic time it takes a large number of swarm sensors to detect a leak

5.2 Data Analysis

By changing the number of agents in the environment and the distance in which they sense blocks, these three characteristics of the system can be changed:

1. The number of agents simultaneously working to complete the structure
2. The available space in the environment for the agents to perform tasks
3. The time agents spend searching for blocks vs. the time agents spend building the structure

One of the key questions trying to be answered is at what point does adding additional agents no longer greatly benefit the construction time. By gradually increasing the number of agents, the time to complete the structure is reduced as shown in Figure 17 and Figure 18. It can be seen that completion time for the structure decreases rapidly up to a point of about 10 agents. After this point, returns for additional agents begin to diminish and the completion time for the structure approaches an asymptotical limit. It might be rationalized that the usefulness of additional agents is directly related to both the size of the environment and the size of the structure that is being built. Eventually, the agents should tax the available space for the environment and completion time for the structure should go up. But, this limit is never reached for the test case due to computing constraints and lack of a parallel implementation. It is expected this behavior will occur after the number of agents exceeds the number of building blocks in the system.

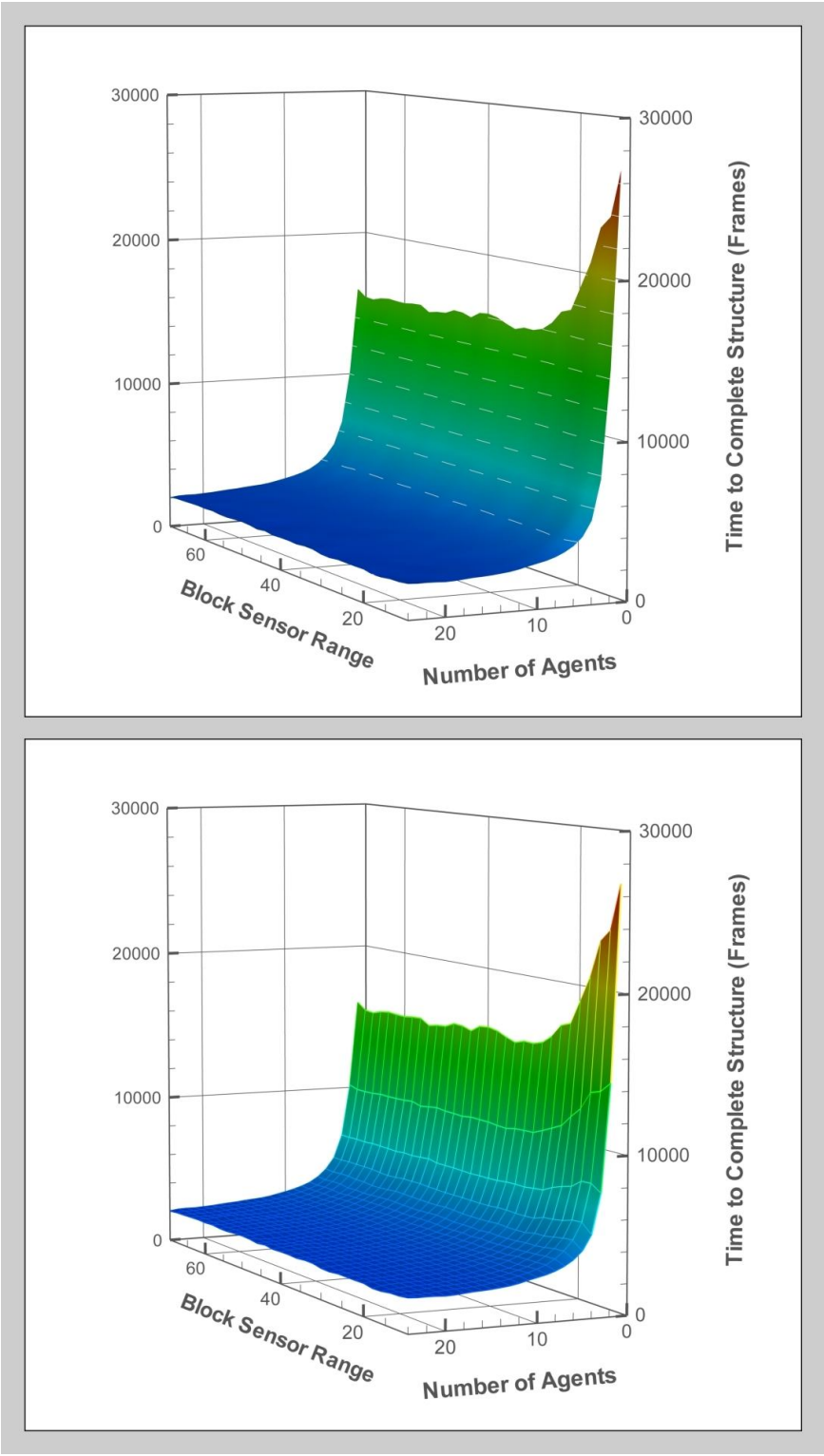

Figure 17: (Top) Contour plot for the sensor test case. (Bottom) Contour plot with mesh data for the sensor test case.

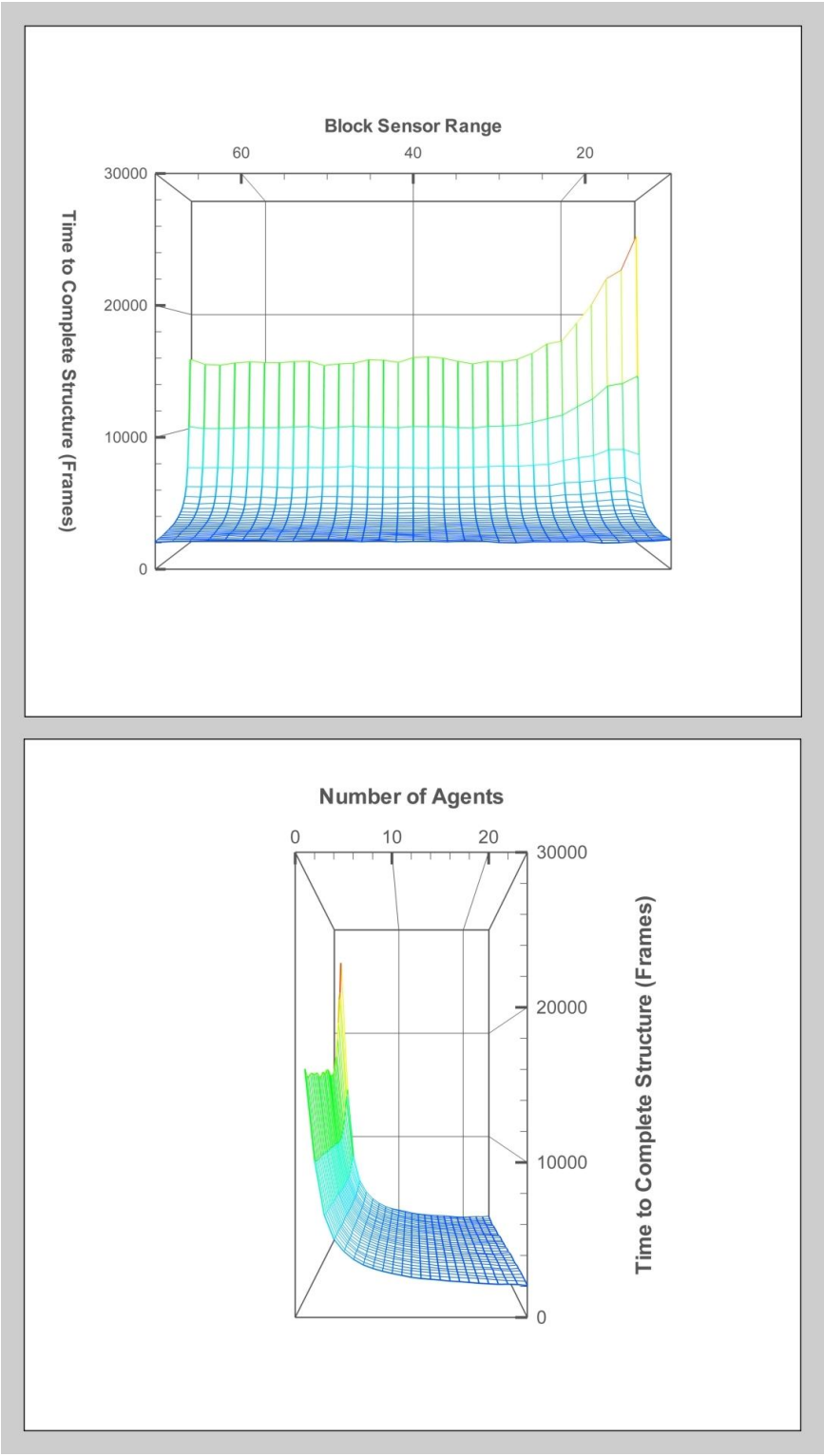

Figure 18: (Top) Side view of "Blocks Sensor Range vs. Time". (Bottom) Side view of "Number of Agents vs. Time".

The second parameter varied is the range of the *block sensor* for the agents. From Figure 17 and Figure 18, it is evident that range capability is much more important to the performance of the system when the number of agents is small. As the number of agents increase, the range of the *block sensor* becomes less influential. This makes sense due to the distributed nature of the agents. An interesting observation for the experiments dealing with 3, 4, and 5 agents is that an initial increase in sensing capability actually reduces the overall performance of the system (see Figure 18). Analysis such as this could be useful in finding idiosyncrasies that are not inherently obvious for a particular control scheme and application.

The functional relationship discussed in section 5.1 represents an exponential decay function. This function was fit to the experimental data from one set of simulations for the sensor test case. The two curves for both are shown below in Figure 19. It can be seen, the control effectiveness for the *Communicating Blocks* algorithm is very similar to that for the swarm system analysis in [Fronczek and Prasad, 2007].

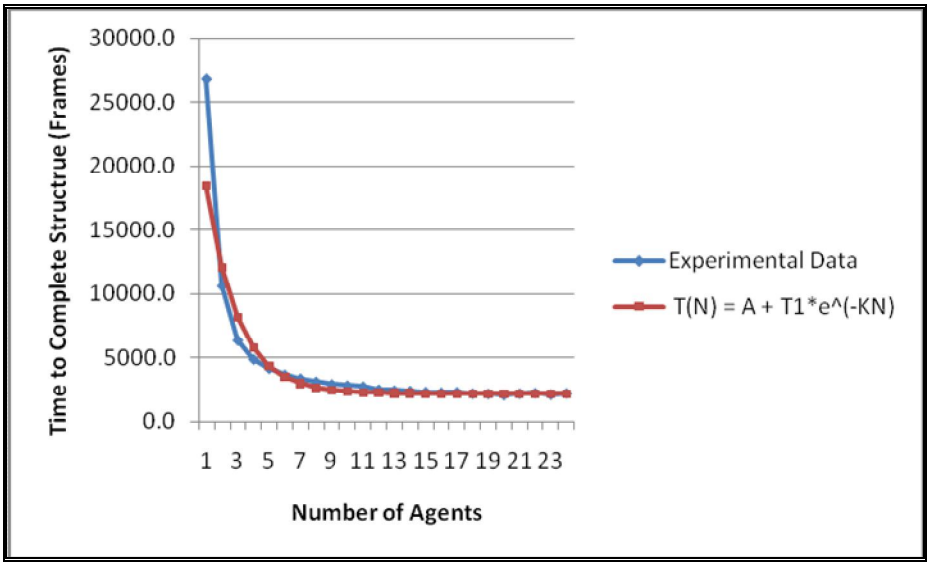

Figure 19: Possible functional relationship for the construction system.

This test case shows the possibility of a virtual simulation in VE-Suite to explore different performance parameters for a given control algorithm. This will allow many algorithms to be tested and classified based on the mathematical relationships they exhibit. Such a functional relationship for control algorithms would be highly valuable when designing new sensor systems. An engineer would be able to decide what kinds of sensing capabilities are required to simultaneously achieve the most cost-effective solution and provide the required value of information for the system.

CHAPTER 6. CONCLUSIONS

As sensors become more widely available, a methodology will be needed to address the following questions: Where should the next sensor be located? What is the value of the information the next sensor will provide? How will the sensors be controlled to work on a collective task? How will the sensor data fidelity be verified? How will the sensor data facilitate the decision making process?

There is still much work that needs to be completed for linking multiple disparate data streams: human interaction, sensor data, and model data. Specifically, control algorithms that process this massive amount of information for a system need to be developed so they can be analyzed in conjunction with real engineering systems. This paper discusses the creation of a virtual framework in VE-Suite and its application to the development of control schemes for large sensor networks. A test case is created to validate the usefulness of a simulated environment in determining potential control algorithms for autonomous sensor networks. Although there are still many aspects of interactive design this framework has not addressed, it is shown this design architecture can be effective in developing the mathematical understandings, algorithms, and control strategies needed to utilize massive sensor systems. Further experiments using this framework are needed to define other aspects of sensor performance.

6.1 Future Research

The construction setting discussed will be used to investigate different informational algorithms, like *Communicating Blocks*, to find what control schemes work well for autonomous systems. Other relationships need to be explored in the framework using

different structures and environmental configurations. In particular, it might be interesting to create a heterogeneous mix of agents with different capabilities and control schemes to see what types of emergent behavior can be accomplished.

Other work will focus on different approaches to the problem. As the mathematical understandings and algorithms are developed, an open-source library of tools will need to be developed that can implement these algorithms in software and hardware components. These components can then be hooked into VE-Suite using the VE-Open specification to create a real sensing application. VE-Open provides the middleware necessary that will allow for both distributed and self-organizational schemes to be implemented. Algorithms will run on local hardware for each component creating an autonomous node that is intelligent and aware of other models in the data structure.

The overall goal is to create a collection of tools that can synergistically be used to facilitate the process of designing new sensing systems. The framework can then be applied to create novel solutions in effective data management and flow for distributed sensing and control systems.

6.1.1 Sensor Network of 48 Sensors

The construction of a sensor network has begun that includes 6 compute nodes (Apple's Mac Minis) with 2 processors per node and 8 channels of sensor data per node. The nodes have been networked together to allow communication of local information from the 8 channels of sensor data. With the sensor network running and measuring data, the sensor data will be displayed real-time in VE-Suite.

The compute nodes will have streaming sensor data from common and different spatial locations, which will require self-organization algorithms to be employed to process the sensor data. With the processing power available with each node, parallel computation is possible.

6.1.2 Hybrid Fuel Cell Power Generation Facility

The tools developed to explore the surrogate autonomous construction agent problem will be applied to controlling information from a hybrid fuel cell power generation facility (see Figure 20) [Tucker et al., 2005]. Although the facility has relatively few sensors (40–100), the platform will provide a pragmatic application to explore the information control schemas developed with the construction agents. In this plant, with 40-100 sensors, it will be possible to explore “what if” scenarios with the live streaming data and the frameworks described in the research by sending the information from the plant to the agents to access. The information streaming from the hybrid plant will then be displayed in the virtual environment based on the control schema implemented. Using the surrogate problem should allow many more information control patterns to be explored before working directly with the power facility. In addition, the surrogate problem can prevent costly failures that may result from using the experimental power facility for exploring information control schemes.

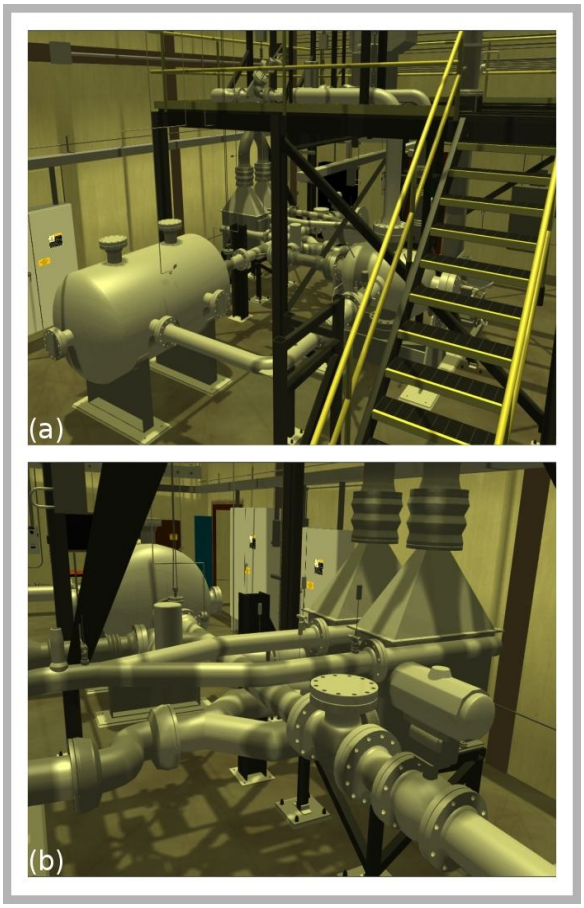

Figure 20: Real-time rendering in VE-Suite of the experimental hybrid fuel cell turbine power generation facility.

APPENDIX. STATISTICAL RESULTS

Sensor Test Case Simulation Results. The average amount of time in frames to successfully complete the structure for each variation of “Number of Agents” and “Block Sensor Range” during the simulation runs

	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38
1	26823.7	23597.4	23316.6	20731.0	19231.9	17459.3	17472.9	16512.2	16003.6	15798.7	15883.9	15580.5	15853.9	16156.5	16273.3
2	10635.6	11021.0	10940.2	10133.0	9965.0	9356.7	9019.8	8976.5	8548.8	8778.4	8637.4	8474.4	8378.7	8414.7	8291.4
3	6381.7	7165.8	7327.9	6658.9	6673.5	6676.4	6020.1	6204.2	6174.2	6128.9	6064.0	6103.6	6059.1	5977.7	5939.7
4	4899.1	5603.0	5268.2	5258.9	5102.4	5168.9	4956.0	4765.9	4882.6	4843.6	4955.2	4777.4	4781.4	4852.7	4838.3
5	4188.5	4636.7	4530.8	4495.2	4290.6	4178.8	4107.3	4120.6	4071.7	4208.6	4088.9	4064.9	4038.8	4022.4	4153.7
6	3692.9	4122.1	4031.1	3736.3	3739.1	3884.4	3653.4	3700.5	3679.9	3696.6	3564.9	3639.8	3674.6	3630.5	3675.3
7	3394.0	3538.5	3607.4	3571.0	3463.5	3400.0	3474.2	3352.0	3272.1	3401.8	3243.8	3406.9	3284.2	3290.4	3293.4
8	3137.2	3362.0	3178.4	3297.5	3294.3	3079.8	3078.9	3073.9	3114.0	3105.5	3092.9	3134.7	3060.6	3050.2	3129.2
9	2945.8	3072.3	3067.4	2928.5	3006.7	2947.8	2899.5	2989.2	2911.5	2916.7	2886.7	2894.3	2964.6	2873.2	2939.0
10	2848.0	2924.9	2854.0	2756.0	2871.8	2848.7	2803.1	2832.6	2767.9	2773.4	2790.5	2758.6	2757.9	2861.5	2786.2
11	2762.6	2789.9	2788.6	2668.6	2627.2	2789.8	2688.5	2746.7	2604.4	2713.6	2710.6	2612.4	2716.1	2723.4	2678.4
12	2533.0	2547.7	2552.0	2624.8	2672.5	2573.1	2619.1	2592.9	2571.5	2597.0	2619.7	2652.1	2547.0	2493.1	2640.7
13	2456.5	2550.6	2457.7	2432.3	2372.5	2567.6	2510.6	2432.2	2418.3	2521.7	2491.0	2469.3	2541.2	2471.6	2525.5
14	2418.3	2504.1	2409.2	2411.5	2402.9	2444.4	2437.0	2385.4	2471.6	2400.0	2387.5	2533.7	2513.3	2513.9	2368.3
15	2340.5	2338.4	2364.0	2422.0	2254.5	2344.9	2325.5	2394.9	2338.8	2465.3	2409.3	2396.5	2515.5	2418.7	2339.0
16	2290.2	2324.3	2335.8	2283.8	2342.1	2329.9	2278.1	2384.2	2354.8	2435.4	2364.8	2320.2	2328.9	2285.0	2307.7
17	2331.1	2285.6	2239.6	2343.5	2221.5	2261.1	2334.2	2253.5	2282.7	2308.9	2249.5	2392.3	2379.7	2299.2	2311.8
18	2219.6	2291.2	2364.5	2183.2	2158.8	2227.4	2263.6	2169.4	2304.7	2253.6	2315.5	2295.8	2232.1	2340.8	2210.5
19	2197.2	2260.4	2258.5	2154.5	2157.8	2139.3	2237.3	2256.6	2148.2	2193.1	2132.9	2155.3	2340.0	2219.9	2296.2
20	2116.0	2245.1	2197.0	2223.1	2200.1	2118.6	2122.7	2187.4	2162.8	2233.5	2184.8	2214.0	2138.2	2300.6	2202.0
21	2189.6	2140.3	2154.1	2177.7	2120.8	2201.3	2155.7	2160.6	2110.1	2083.0	2168.9	2196.0	2218.8	2184.9	2095.0
22	2258.5	2149.9	2066.7	2131.9	2144.9	2205.5	2150.3	2135.4	2114.1	2170.5	2153.9	2209.6	2206.7	2115.2	2163.2
23	2152.0	2061.6	2130.3	2065.9	2164.7	2083.9	2102.7	2141.9	2043.9	2015.6	2057.0	2203.8	2095.4	2033.3	2151.4
24	2242.3	2136.0	2131.9	2009.8	1976.3	2122.4	2082.6	2086.9	2101.8	2000.8	2020.3	2164.8	2168.6	2073.6	2132.2

	40	42	44	46	48	50	52	54	56	58	60	62	64	66	68	70
1	16231.4	15687.0	15984.0	16035.3	15658.3	15649.4	15425.7	15918.7	15835.0	15722.4	15751.3	15820.2	15725.0	15489.6	15539.4	16033.5
2	8444.1	8478.7	8415.0	8321.9	8933.6	8628.5	8511.5	8726.3	8545.7	8587.1	8443.8	8486.6	8296.3	8462.2	8345.7	8569.4
3	5926.4	6129.6	5996.8	5968.3	6062.0	6054.8	5957.2	5968.6	5972.9	5963.3	6014.5	6033.6	6022.3	6049.3	6096.8	5888.4
4	4785.9	4655.8	4840.6	4890.4	4882.8	4808.6	4883.4	4783.4	4640.9	4802.1	4887.1	4895.9	4796.2	4842.9	4722.5	4839.6
5	4201.4	4092.9	4084.7	4091.0	4101.6	4188.0	4212.8	4035.6	4059.3	3964.9	4091.1	4041.9	4222.8	4193.6	4143.9	4219.4
6	3721.0	3710.5	3584.4	3618.3	3604.5	3642.2	3629.1	3662.7	3608.9	3507.4	3603.8	3568.1	3690.7	3700.0	3621.6	3697.8
7	3356.7	3390.5	3330.4	3285.2	3347.8	3375.5	3376.2	3339.5	3344.0	3335.0	3349.4	3188.8	3327.7	3344.4	3273.3	3331.0
8	3113.9	2991.0	3126.1	3121.8	3144.3	3126.3	3092.5	3086.6	3207.6	3232.3	3187.1	3084.1	3154.0	3108.9	3142.6	3172.8
9	2834.9	2796.3	2876.3	2998.1	2961.6	2890.0	2947.9	2946.2	2905.5	2932.5	2878.1	2919.4	2907.9	3027.3	2937.4	2963.0
10	2757.9	2839.3	2720.6	2804.6	2918.9	2815.4	2797.8	2844.3	2821.3	2768.5	2893.5	2787.9	2845.7	2683.5	2795.6	2833.7
11	2641.4	2720.1	2725.5	2578.1	2613.9	2575.3	2745.3	2700.3	2648.2	2587.9	2731.3	2677.1	2684.4	2668.7	2748.7	2739.0
12	2601.2	2532.4	2554.7	2615.8	2642.6	2592.0	2542.2	2636.1	2630.3	2577.4	2586.7	2529.3	2495.7	2635.1	2542.9	2552.8
13	2517.6	2542.1	2534.8	2497.7	2516.2	2484.2	2469.4	2360.5	2486.5	2446.9	2477.3	2531.6	2509.2	2543.8	2491.7	2507.7
14	2561.0	2453.3	2455.9	2364.3	2395.6	2424.5	2475.6	2416.8	2480.6	2424.3	2470.9	2421.3	2479.6	2458.6	2437.2	2446.2
15	2374.8	2365.0	2497.1	2407.3	2433.6	2318.5	2430.9	2324.5	2378.2	2335.5	2325.3	2314.2	2418.0	2426.5	2407.6	2467.1
16	2319.0	2325.5	2285.0	2347.2	2428.6	2388.6	2371.8	2455.9	2245.5	2449.7	2266.0	2351.5	2405.6	2404.2	2407.3	2278.4
17	2276.5	2268.3	2250.8	2315.7	2361.1	2230.3	2282.2	2326.7	2294.5	2429.1	2234.2	2243.1	2283.4	2324.3	2298.1	2356.4
18	2173.5	2241.2	2276.9	2214.6	2190.1	2240.2	2338.9	2279.6	2354.1	2288.4	2353.8	2321.4	2296.3	2306.2	2347.0	2264.6
19	2260.5	2119.6	2254.9	2282.0	2222.7	2314.8	2216.8	2264.9	2355.6	2245.8	2238.3	2237.7	2170.9	2262.1	2310.2	2159.2
20	2133.2	2406.8	2243.4	2111.9	2146.4	2185.1	2267.4	2272.7	2166.6	2319.6	2235.0	2244.0	2260.8	2117.1	2271.6	2225.2
21	2268.4	2228.9	2127.2	2197.0	2072.8	2280.4	2232.0	2076.3	2299.4	2142.0	2192.7	2338.8	2130.1	2170.2	2182.1	2057.4
22	2143.6	2197.3	2114.6	2127.1	2048.8	2149.7	2127.8	2185.9	2153.2	2229.6	2064.5	2169.8	2126.5	2114.5	2151.3	2189.0
23	2224.1	2153.6	2133.3	2127.9	2086.1	2096.9	2133.0	2253.0	2098.0	2025.3	2102.6	2136.7	2114.9	2215.3	2146.7	2162.1
24	2121.7	2069.8	2198.3	2069.9	2184.7	2217.0	2180.5	2120.9	2125.1	2208.8	2150.7	2187.8	2160.8	2127.6	2117.2	2047.5

REFERENCES

ACE and TAO, Adaptive Communication Environment and The ACE Orb, Retrieved November 9, 2008, from http://www.cs.wustl.edu/~schmidt/TAO.html/.

Argo, part of the integrated global observation strategy, Retrieved November 9, 2008, from http://www-argo.ucsd.edu/.

Bonabeau, E., G. Theraulaz, J-L. Deneubourg, N.R. Franks, O. Rafelsberger, J-L. Joly, and S. Blanco. (1988). A Model for the Emergence of Pillars, Walls and Royal Chambers in Termite Nests, *Philosophical Transactions of the Royal Society of London*, vol. 353, pp. 1561-1576.

Borenstein, J. and Y. Koren. (1989) Real-time obstacle avoidance for fast mobile robots, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179-1187.

Broten, Greg, Simon Monckton; Jared Giesbrecht & Jack Collier. (2006). Software Systems for Robotics: An Applied Research Perspective, *International Journal of Advanced Robotic Systems*, vol. 3, pp. 11-17.

Bullet Physics Library, Retrieved November 9, 2008, from http://www.bulletphysics.com/.

Carmen Robot Navigation Toolkit, Retrieved November 9, 2008, from http://carmen.sourceforge.net/.

Chong, Chee-Yee, S.P. Kumar, and Booz Allen Hamilton. (2003). Sensor networks: evolution, opportunities, and challenges, *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247-1256.

CORBA, Common Object Request Broker Architecture, Retrieved November 9, 2008, from http://www.corba.org/.

Cote, Carle, Dominic Letourneau, Francois Michaud, Jean-Marc Valin, Yannick Brosseau, Clement Raievsky, Mathieu Lemay, Victor Tran. (2004). Code Reusability Tools for Programming Mobile Robots, *Proceedings of the IEEE International Conference on Intelligent Robots and System*, vol. 2, pp. 1820-1825.

Das, SK, N. Banaerjee, and A. Roy. (2006). Solving Optimization Problems in Wireless Networks using Genetic Algorithms, in *Handbook of Bioinspired Algorithms and Applications* (ed. Olariu S, Zomaya AY and Olariu O) CRC Press.

Di Caro, Gianni, Frederick Ducatelle, and Luca Maria Gambardella. (2005). AntHocNet: An Adaptive Nature-Inspired Algorithm for Routing in Mobile Ad Hoc Networks, European Transaction on Telecommunications, Special Issue on Self-organization in Mobile Networking, vol. 16, pp. 443-455.

Dressler, Falco. (2008). Self-Organization in Sensor and Actor Networks, John Wiley & Sons, Ltd.

Forward Based X-Band Transportable (FBX-T) Radar, AN/TPY-2/TPS-X/Forward Deployable Radar (FDR), Retrieved November 9, 2008, from http://www.globalsecurity.org/space/systems/fbx-t.htm/.

Fronczek, J.W., and N.R. Prasad. (2005). Bio-inspired sensor swarms to detect leaks in pressurized systems, *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, pp. 1967-1972.

Gerkey, Brian P., Richard T. Vaughan, Kasper Støy, Andrew Howard, Gaurav S. Sukhtame, and Maja J. Matarić. (2001). Most valuable player: A robot device server for distributed control, in *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS at Autonomous Agents 2001*, Wailea, Hawaii, pp. 1226-1231.

Gerkey, Brian P., Richard T. Vaughan, and Andrew Howard. (2003). The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems, in *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, pp. 317-323.

Grassé, P-P. (1959). La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis* et *Cubitermes* sp. La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, vol. 6, pp. 41-81.

Haenselmann, Dr. Thomas. (2006). *Lecture on Sensor Networks*. University of Mannheim, Germany. Retrieved November 9, 2008 from http://www.informatik.uni-mannheim.de/~haensel/sn_book/slides/motivation.pdf/.

Qi, Hairong, S. Sitharama Iyengar, Krishnendu Chakrabarty. (2001). Multiresolution data integration using mobile agents in distributed sensor networks, *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 31, no. 3, pp. 383-391.

Holland, Owen and Chris Melhuish. (1999). Stigmergy, Self-Organization, and Sorting in Collective Robotics, in *Artificial Life*, vol. 5, no. 2, pp. 173-202.

Koenig, Nathan and Andrew Howard. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, vol. 3, pp. 2149-2154.

Kruger, Daniel, Ingo van Lil, Niko Sunderhauf, Robert Baumgartl, and Peter Protzel. (2006). Using and Extending the Miro Middleware for Autonomous Mobile Robots, Chemnitz University of Technology: Department of Electrical Engineering and Information Technology, Chemnitz, Germany.

Kuo, Yuan-hsin and Bruce A. MacDonald. (2004). Designing a distributed real-time software framework for robotics, in *Proceedings Australasian Conference on Robotics and Automation*, Canberra, Australia.

Mainwaring, Alan, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. (2002). Wireless Sensor Networks for Habitat Monitoring, in WSNA,

MARIE, Retrieved November 9, 2008, from http://marie.sourceforge.net/mediawiki/index.php/Main_Page/.

McCorkle, Doug, Chngguan Yang, Terry Jordan, Dave Swensen, Stephen E. Zitney, and Mark Bryden. (2007). Toward the Integration of APECS with VE-Suite to Create a Comprehensive Virtual Engineering Environment, *32nd International Technical Conference on Coal Utilization & Fuel Systems*, Clearwater, FL, June 2007.

Michael, Nathan, Jonathan Fink, and R. Vijay Kumar. (2008). Experimental testbed for large multi-robot teams: Verification and validation, *IEEE Robotics and Automation Magazine*.

Montemerlo, Michael, Nicholas Roy, and Sebastian Thrun. (2003). Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit, in *Proceeding of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, 2436-2441.

(The) Orocos Project, Retrieved November 9, 2008, from http://www.orocos.org/.

(The) Player Project, Retrieved November 9, 2008, from http://playerstage.sourceforge.net/.

Photo of Giant Termite Mounds in Litchfield National Park, Retrieved November 9, 2008, from http://www.tripadvisor.com/ReviewPhotos-g494971-d256773-r6022892-Litchfield_National_Park-Batchelor_Top_End_Northern_Territory.html/.

Romer, Kay, and Friedemann Mattern. (2004). The design space of wireless sensor networks, in *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54-61.

Sherwood, R., S. Chien, D. Tran, B. Cichy, R. Castano, A. Davies, G. Rabideau. (2006). Autonomous science agents and sensor Webs: EO-1 and beyond, *Aerospace Conference, 2006 IEEE*, pp. 10.

Teller, Seth. (2005). Guide to the Use of the Carmen Mobile Robot Control Package Within RSS, Retrieved November 9, 2008, from http://courses.csail.mit.edu/6.141/spring2008/pub/labs/VisualServo/docs/Carmen-Guide.html/.

Theraulaz, Guy, and Eric Bonabeau. (1999). A Brief History of Stigmergy, *Artificial Life*, vol. 5, no. 2, pp. 97-116.

Tucker, David, Larry Lawson, and Randall Gemmen. (2005). Characterization of Air Flow Management and Control in a Fuel Cell Turbine Hybrid Power System Using Hardware Simulation, *2005 ASME Power Conference*, Chicago, IL.

VE-Suite, Virtual Engineering Suite, Retrieved November 9, 2008, from http://www.vesuite.org/.

Wawerla, James, Gaurav S. Sukhatme, and Maja J. Matarić. (2002). Collective Construction with Multiple Robots, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, pp. 2696-2701.

Werfel, Justin, Y. Bar-Yam, D. Rus, and Radhika Nagpal. (2006). Distributed Construction by Mobile Robots with Enhanced Building Blocks, *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pp. 2787-2794.

Werfel, Justin, and Radhika Nagpal. (2006). Extended Stigmergy in Collective Construction, *IEEE Intelligent Systems*, vol. 21, no. 2, pp. 20-28.

Xiao, Angran, Kenneth Mark Bryden, and Douglas S. McCorkle. (2005). VE-Suite: A Software Framework for Design-Analysis Integration During Product Realization, *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 3, Pts A and B, pp. 859-867.

Zhao, Feng, and Leonidas J. Guibas. (2004). Wireless Sensor Networks: An Information Processing Approach, Elsevier/Morgan Kaufmann. San Francisco, CA.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Mark Bryden for his direction, insight, and wisdom in my academic career thus far. I would like to thank Dr. Terry Meyer and Dr. Carolyn Heising for taking the time to participate on my committee. I would like to thank each member of the Virtual Engineering Research Group for their contributions, especially Doug McCorkle for his assistance and guidance at the early stages of this research, and also Adam Shuttleworth for continuing research related to this project.

I would like to thank my parents, Bruce and Kristy, for their love and support, and for encouraging me to pursue my education at Iowa State University. Finally, I would like to thank my girlfriend Angie for her love and support, and also her understanding on the weekends while I was writing my paper.

This research work at the Ames Laboratory was supported by the Department of Energy-Fossil Energy under Contract No. DE-AC02-07CH11358.