Infotech@Aerospace 2012
19 - 21 June 2012, Garden Grove, California

AIAA 2012-2568

A Lego Mindstorms NXT-Based Test Bench for Cohesive Distributed Multi-agent Exploratory Systems: Mobility and Coordination

Noah Maze*, Yan Wan†, Kamesh Namuduri‡, and Murali Varanasi§
University of North Texas, Denton, TX, 76201

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

A cost-effective and time-saving test bench can significantly help with the study and implementation of distributed multi-agent exploratory systems. In this paper, we introduce a computer-based test bench that controls multiple low-cost Lego Mindstorms NXT programmable robotics kits via Bluetooth. We identify mobility and coordination as the two critical components to establish a platform for exploratory systems, and describe in detail how they are implemented using the low-cost Lego Mindstorms robots. In particular, the test bench includes a variety of ground-based or airborne mobility models, such as the Random Direction, Random Way-point, Flight Plan-based, and Smooth-turn mobility models. It also includes a distributed task assignment algorithm, based upon the stochastic automaton known as the influence model. At the end of the paper, we present a search-and-rescue example to illustrate the use of the test-bench. We expect that the test bench can serve as a conduit to foster the real-world application of distributed multi-agent exploratory system concepts.

I. Introduction

The delay of round-trip communication from Earth to Mars or from the ground to the air complicates the teleoperation of unmanned vehicles tremendously. A solution to minimizing the burden on teleoperation is to make the system of unmanned vehicles more autonomous. Military and civilian applications such as reconnaissance, patrol and space exploration also demand more robust exploratory systems. Distributed multi-agent systems are envisioned to meet both of these requirements, but the design, implementation, and evaluation of high-performance distributed mechanisms under a variety of physical constraints is not trivial.

We have developed a Lego Mindstorms NXT-based test bench that could help engineers overcome this challenge by providing a low-cost, flexible and easy-to-implement foundational architecture on which we can implement and evaluate various distributed mechanisms for real-world multi-agent exploratory systems. Specifically, the key advantages of this test bench are that it: 1) streamlines the initial test bench creation process, while giving engineers unfettered access to the actuators, sensors and communications systems of the agents; 2) dramatically reduces the time between conception and prototyping of a multi-agent system design; 3) offers the key strength of software simulations: versatile agents can be reprogrammed and redesigned with ease, and most importantly 4) provides a lower price-point but full-fledged physical realization that facilitates a level of interaction with the system that is not feasible within the confines of a conventional software simulation.

*Undergraduate Student, Department of Electrical Engineering, and AIAA Student Member.
†Assistant Professor, Department of Electrical Engineering, and AIAA Member.
‡Associate Professor, Department of Electrical Engineering.
§Professor and Chair, Department of Electrical Engineering.

Copyright © 2012 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

This paper documents the implementation of the test bench, in particular how the low-cost in-laboratory platform mimics and realizes real-world exploratory applications. We discuss a variety of issues in communication and perception, but largely focus on two critical areas: mobility and coordination. The remainder of this section briefly introduces the two focus areas to be discussed and implemented in the paper. We envision that this test bench will serve as a conduit to foster the real-world application of distributed multi-agent exploratory system concepts.

Random Mobility Mobility models serve as the foundations for the implementation and evaluation of communication strategies and routing protocols [6]. Effective simulation of mobility must realistically reflect the constraints placed on real-world agents and provide engineers with meaningful information about the movement of each agent. These two goals are met by providing various drive models and analyzing recorded agent movement.

The test bench supports drive models that can be controlled by a pre-trip exploration plan or by a human via teleoperation, but we are particularly interested in randomized exploration. End users have access to abstracted mobility models that simplify the act of exploration. The choices include conventional two-dimensional random mobility models such as random direction (RD) [10] and random way-point (RWP) [3], which describe the movement of ground-based rovers. We also consider the simulation of autonomous airborne exploration. Because of the high-cost of airborne vehicles, establishing a ground-based platform to emulate airborne networks is of significant value. In particular, the end users can choose more advanced drive concepts that mimic the characteristics of air travel: flight plans [12] and our newly developed smooth-turn mobility model [13].

Distributed Coordination The autonomy of each individual agent is largely based on the mobility and sensing portions of the multi-agent system, but in order to have the robots accomplish more complicated tasks, coordination among multiple agents is needed. As opposed to centralized coordination where the central command node represents a point of vulnerability that could ruin an entire mission, decentralized networking offers the most robust communication strategy to uncertain environments and malicious attacks [9].

We implement and test several distributed task assignments in the test bench. In the circumstance that multiple tasks are present, it is a practical problem for the agents to partition among themselves into groups to complete the tasks simultaneously. We implement a stochastic-automaton-based partitioning algorithm [14] for this task assignment problem.

The rest of the paper is organized as follows. In Section II, we describe the theoretical background of the mobility models and the distributed task assignment. Section III includes the details of the implementation using Lego Mindstorms NXT robots in the Labview environment. In Section IV, we describe the results and also demonstrate a search-and-rescue example to illustrate the use of the test bench for realistic applications. In the example, the robots randomly search for multiple targets, partition themselves in an entirely distributed fashion into multiple groups, and swarm toward the targets to attend to them. Finally Section V includes a brief conclusion.

II. Theoretical Background

In order to create a test bench that is as accessible and useful as possible, we include a variety of mobility models, that are widely used as the foundation for networking and communication studies. Decentralized decision making is carried out on-the-fly based on the proximity and state of nearby agents. In this section, we describe the theoretical fundamentals of these models, with the focus on their applicabilities, model details, and properties.

A. Mobility Models

Effective information routing design for mobile networks has been one of the key areas of research in communication and networking. The routing design is significantly complicated by *time-varying* network structure

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

caused by the uncertain movement of network agents. As such, mobility models which abstract the random movement patterns of network agents serve as the foundation and analytical framework for the evaluation and design of routing protocols [5, 6].

The mobility models we considered can be broken down into two categories. Typical ground-based exploration models (such as RD and RWP) are implemented first, and then our focus is shifted to more advanced models (Flight-plan based, Smooth-turn), that emulate the constraints of aerial vehicles. Different from ground vehicles which are capable of making sharp stops and turns, gliding vehicles tend to maintain a constant speed but show down through zigzagging, and change direction through making large turns. These advanced models create randomized exploration while appropriately capturing the kinematics of gliding vehicles, in particular the temporal and spatial correlation of vehicle movement. The details of the four mobility models are described in the following. The test bench keeps track of the agents' positions as they move.

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

RD Model The simplest RD model explores a predefined area by choosing a random direction Φ from a uniform distribution (i.e., the probability distribution $f(\Phi) = \frac{1}{2\pi}$) and an exponentially distributed traveling duration T (such that $f(T) = \lambda e^{-\lambda_d T}$, where λ_d is the mean duration) [10]. The exponential distribution is typically used to capture the waiting time between arrivals, and featured by its nice memory-less properties that facilitates analysis [11]. After completing the duration, the agent randomly chooses another direction and duration. The agents reflect off of the bounds of the map when they are encountered. This mobility leads to a uniform distribution of stationary node locations [10], which further leads to a variety of analytical results on network connectivity, such as the statistical number of neighbors for each agent, and the probability for a $(k-)$ connected network [2, 15].

RWP Model The RWP model addresses a different sort of random exploration: the agent chooses a uniformly distributed random location within the playing field, and navigates towards it, and pauses there before it chooses another destination [3]. For instance, for a rectangular area $L \times W$, the probability for the x and y location of each destination is $f(x) = \frac{1}{L}$, and $f(y) = \frac{1}{W}$. Moreover, the pause time follows also exponential distribution with mean λ_p. Despite the similarity between RD and RWP models, RWP model demonstrates a non-uniform node distribution, for which the close-form representation has not been obtained in the literature [3]. The difference between the two models is caused by the boundary effect.

Flight-plan based Model The simplest mobility model for airborne networks is based on pre-defined flight plans [12]. The flight plan is converted to topology maps, which are updated to accommodate deviations from the plan. The flight-pan based model is well suited for transportation purposes where the source and the destination are determined. Comparing to the other mobility models, this model has the most predictability for future trajectories. Using this model for exploratory systems requires the entire flight plan to be generated beforehand, and stored on each agent. This could create an exploitable vulnerability and it requires more resources than a randomly determined route.

Smooth-Turn Model Recently, our group developed a smooth-turn mobility model, that provides a foundation for random, spontaneous mobility that is compatible with the constraints of flight and minimizes the amount of locational information stored within the agent [13]. The model resembles the RD model, but capturing the preference of flights toward making straight and smooth-turn trajectories. Also, like the RD model, the agents must know the bounds of the area to be explored, but further information about the planned flight of the agents will be generated ephemerally. The model works as follows: each agent randomly chooses a turning center perpendicular to his heading angle, and move around this center for an exponentially lasted duration with mean λ_s, before choosing another turning center (see Figure 1 for a trajectory simulation) [13]. The inverse of the turning radius r follows normal distribution, i.e., $f(\frac{1}{r}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2r^2\sigma^2}}$, where σ is the standard deviation. The normal distribution guarantees the preference toward smooth trajectories and large turns. Moreover, the constraint on the selection of turning center guarantees the smooth trajectory of airborne vehicles. As proved in paper [13], the smooth-turn model

typically has more chaotic network behavior than the aforementioned mobility models, due to the smoothness constraint on trajectories. Moreover, the stationary node distribution is also uniform.

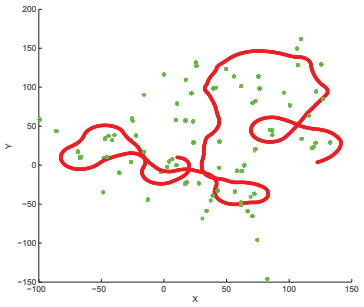

Figure 1: A simulation of the trajectory of UAV in a 2-D domain [13].

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

B. Decentralized Coordination

Ephemeral generation of movement boosts the security of a multi-agent system, but centralized task assignment could eliminate these advantages by broadcasting the intent of the system whenever a decision is made. Decentralized coordination and task assignment allow partitioning decisions to be made on an individual level.

In a search-and-rescue paradigm, multiple targets might present themselves to the swarm simultaneously. Wan et al's network self-partitioning algorithm allows individual units to determine the optimal partition for dividing into teams to address multiple targets, based upon the location adjacency among them [14]. The distributed partitioning algorithm is based upon a stochastic automaton—the copying influence model [1]. The partitioning algorithm has three steps: initiation, iteration, and termination. In the initiation stage, we know an agent associated with each of the groups to start with. These agents are assigned with different states, and the rest of the agents are initialized with random states. The agents also find their relative distances with their neighbors, and translate the distances into probabilities, such that the probabilities from all incoming edges sum to 1. The probabilities are inversely proportional to the relative neighboring distances. In other words, for neighboring agents that are further away from each other, the probability (also known as the influence probability) between the agents is smaller. In the iteration stage, every agent updates its state by copying the state of its neighbors, according to the incoming probability from that neighbor. Moreover, if the states between two neighbors are different, the influence probability is modified by subtracting a small amount δ. The termination stage concludes when the network is partitioned, and the state of every agent does not change. We prove in [14] that in the limit of small δ, the stochastic partitioning algorithm finds the optimal cut with probability 1. Figure 2 demonstrates the initiation, iteration and termination of a simple seven-node network. It should be noted that the branch probabilities alone do not add up to a probability of 1 due to the implied probability that a node will copy itself.

III. Implementation

The Lego Mindstorms NXT programmable robotics kit was chosen as our agent platform due to its accessibility and low price point. At a cost of only $250 per agent, the Lego NXT is a powerful and cost-effective introduction to the basic concepts of robotics, and it has been implemented in many undergraduate courses to facilitate project-based exploration of single-agent control applications [7] [8]. The central processing unit of the Lego NXT is referred to as the NXT Intelligent Brick, and it contains a 32-bit ARM processor, four inputs, three outputs, a basic front panel interface and a Bluetooth module. When NXTs are communicating amongst themselves, they must obey a strict Master/Slave network topology with a maximum of three agents

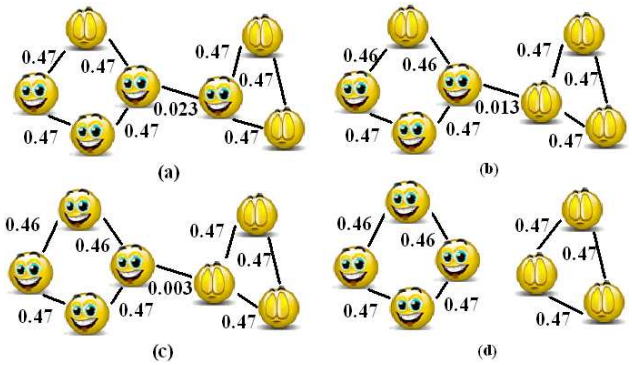

Figure 2: Example illustrating the distributed partitioning algorithm [14]. Smiling face and sad face represent different states.

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

per master and no role-switching; however, communication from other devices to agents is not subject to this limitation.

The agents can be controlled in two different ways: Direct Drive, and Individual Programming. Individual Programming allows each NXT to be completely autonomous. Programs are designed on the NXT itself or within one of many different NXT programming environments, and then uploaded to the Lego NXT as a self-contained program that can be executed immediately or saved for later use. This method is ideal for programming independent agents, but problems arise when multi-agent systems are attempted due to sensor inconsistencies and communication challenges. Brigandi et al's work in multi-NXT localization and cooperation was stymied by the unpredictable nature of NXT sensors and the rigid Bluetooth topology [4].

Direct Drive allows a computer to send commands directly to each Lego NXT for immediate execution in a sequence defined by a series of Sub VIs, abstracted LABVIEW blocks.. This arrangement requires a centralized computer executing a single LABVIEW block diagram, and it eliminates the Bluetooth rigidity: a single LABVIEW instance can interface with the maximum amount of agents the Bluetooth standard will allow. Because of these advantages, we focused on Direct Drive methodology.

A. Basic Movement and Tracking

The mechanical versatility of the Lego NXT system lends itself to a wide range of transportation methods. We chose a basic three-wheeled robot design featuring two independently-driven wheels and a third wheel for stability. This differential drive layout allowed for pure rotation and pure translation: the agents can alter their heading without affecting their location, and vice versa. This simplification turns an under-actuated system in to a sequential pair of much more tractable systems. When the agents are constrained to only pure rotation or pure translation, the localization of said agents reduces to simple trigonometric relationships:

$$\Delta X = d * \arcsin(\theta) \tag{1}$$

$$\Delta Y = d * \arccos(\theta) \tag{2}$$

The magnitude of the distance traveled is d, and θ is the heading, which can be observed to within a degree of accuracy of 1°with the compass sensor. Given the initial position of an agent, the current position can be determined by summing all previous translations. Figure 3 explains how our LABVIEW implementation updates agent location after every translation based on observed heading, and supplied distance. Mobility models that require simultaneous heading and location changes, like the aforementioned models that emulate realistic flight paths, require a more advanced tracking system that will be discussed later.

Once the tracking method was completed, we created Sub VIs for pure rotation and pure translation. A heading adjustment Sub-VI that rotates the agent to the desired heading was implemented first. The

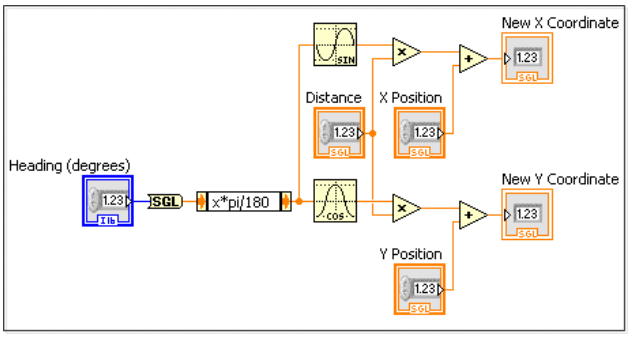

Figure 3: Block diagram describing the Tracking process when rotation and translation are independent

NXT motors do not possess a high level of precision, so a proportional controller was developed that can consistently steer the agent to within 5°of the desired heading (Fig. 4). When a heading is sent to this Sub-VI, the agent adjusts its heading and returns the observed heading after adjustment is complete to maximize the accuracy of the tracking algorithm.

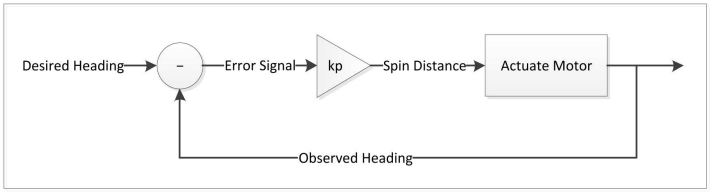

Figure 4: Block diagram describing proportional heading control

A similar proportional controller was designed to provide an accurate forward advance, but the results of its implementation were lackluster. The NXT motors contain sensors that report the angle of rotation; when the motors are reset, the left and right motors are both at angle zero. After a single incremental forward movement, they should theoretically still have equivalent angles. Our controller amplified the difference between the wheels' angles in an effort to maintain this equivalence, but this did not result in a straight line advance because the spin sensor could not detect slippage between the ground and the tire, the primary source of crooked forward movement. As a result, we implemented an unmodified NXT forward movement block instead (Fig. 5).

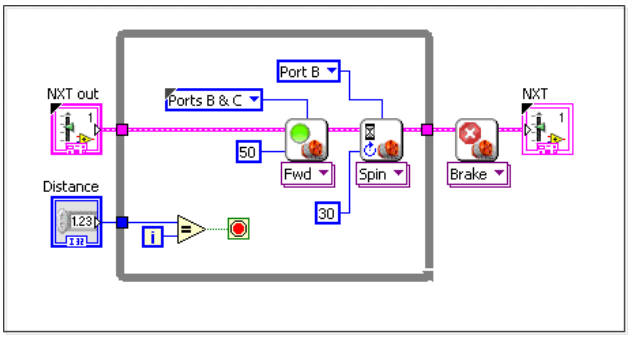

Figure 5: The Forward Movement Sub-VI

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

B. Random Exploration

Once a tracking algorithm and position control were established, we implemented the first mobility model: Random Direction. This mobility model explores an area by randomly choosing a direction and distance, and moving until the edges of the field are encountered. The distribution of these random choices and the behavior of the agent at the boundaries determine the steady-state coverage of the agent.

Uniform randomness is easily created by a foundational block in the LEGO NXT add-on for LABVIEW. The block nominally provides a uniformly distributed integer from 1 to a supplied maximum value. By setting this maximum to 360 and passing the resulting variable to the previously discussed heading controller, the heading changes of the robot become uniformly distributed. A uniform distribution from 1 to the maximum desired distance can be passed to the forward movement Sub VI to provide a similarly uniformly distributed forward advance after each heading change, but most Random Direction implementations require an exponentially distributed forward advance; however, the NXT does not have a native method to generate exponentially distributed random variables. Fortunately, a uniformly random integer distribution can be converted to an exponential one by dividing by the maximum value, taking the natural log, and dividing by $-\lambda$, the scaling factor. The block diagram in Fiure 6 illustrates our implementation of this conversion in the labview environment. The resulting number could potentially be very large if the original random value was very small, so a saturation value was defined to prevent excessively long travel distances.

After the exponentially distributed random number generator was implemented, we logged its output over the course of many iterations. While verifying distribution of the output in MATLAB, a peculiar thing happened: the output was not actually exponentially random. In fact, some numbers were missing altogether! The cause of this problem wound up being a failure of the LEGO NXT Random Integer Block to generate truly uniform random integers initially. The maximum value of this block was extraordinarily high to provide as fine a grain as possible for number generation, but this value was apparently so high that the output was no longer uniformly random. When the maximum value was lowered to the current value of $10,000$ an exponential distribution was observed that met our expectations.

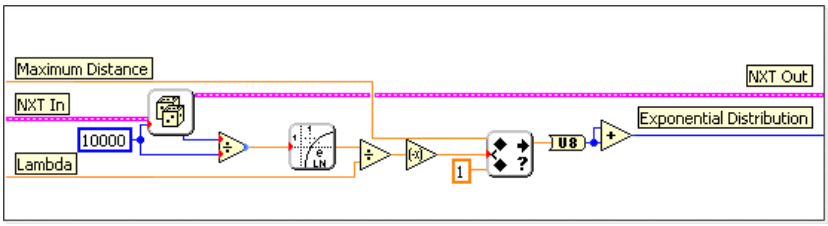

Figure 6: Converting a uniform integer distribution to an exponential distribution

Uniformly distributed heading changes and exponentially distributed distance changes are not quite enough for uniform steady-state coverage. The edges of a finite field of exploration create inconsistencies near the boundaries. In software simulations this problem can be solved by creating a new agent at the center of the field every time one wanders out of bounds, but in a physical implementation this is not possible. Two solutions were considered for this problem: distance saturation and boundary reflection. Distance saturation implies that the robot chooses a random heading and then drives until a boundary is met, and repeats this process ad infinitum. This simple implementation yields a uniform distribution, but the destinations of the robot fall only along the perimeter. Our mobility model (Figure 7) computes the previously discussed distance and heading, and reflects off of the boundaries when they are encountered. The algorithm continues until the remaining distance equals zero, so a large distance input could cause the agent to ricochet off the boundaries for quite some time.

The Random Way-point mobility model (Fig. 8) provides a different sort of uniform distribution: the agent chooses a uniformly distributed random location within the playing field, navigates towards it, and pauses when it arrives. This behavior leads to a uniform distribution of attention and resting locations as pause-time becomes much greater than travel-time, and the boundaries of the space do not cause any navigation problems.

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

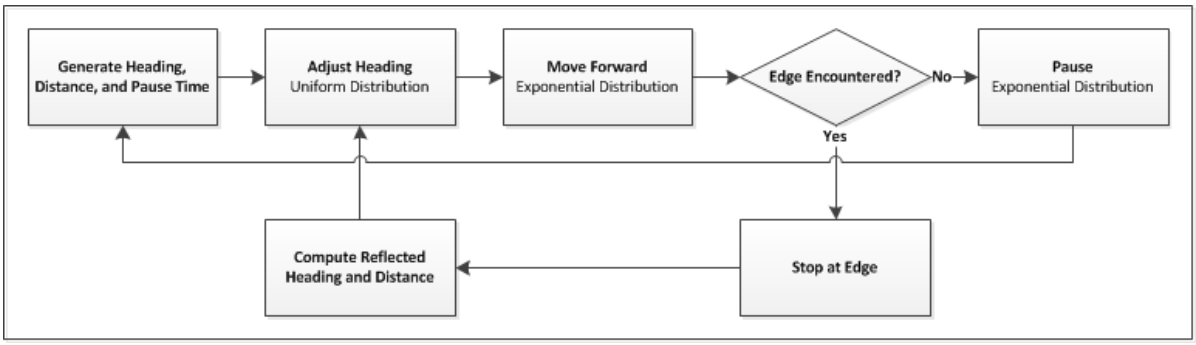

Figure 7: Block diagram of the Random Direction Mobility Model with Boundary Reflection

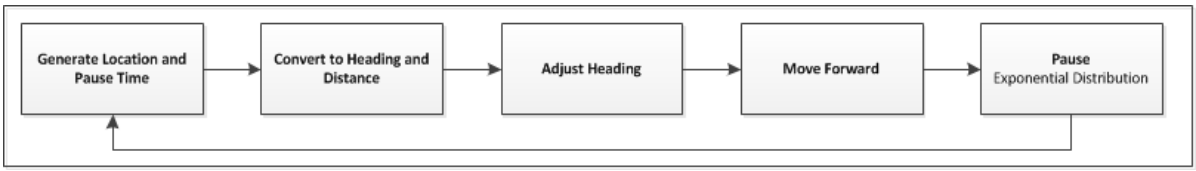

Figure 8: Block diagram of the Random Way-point Mobility Model

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

C. Advanced Movement and Tracking

The ground-based Random Direction and Random Way-point mobility models provide a method for ground-based vehicles and rotor-craft to explore a space, but gliding vehicles are not capable of independent rotation and translation. Flight plans and the smooth-turn mobility model are designed to generate movement in systems that must be constantly on the move.

Instead of basing navigation on pure translation and pure rotation, we had to design a position control that could change both heading and position simultaneously. There are many ways to do this, but we chose to hold velocity constant while controlling the rate of change of the heading of the robot. If the speed of the agent is constant, the rate of change of the heading can be directly mapped to a change in position. Both the Smooth Turn and Flight Plan Movement Models rely on circular paths, so the change in heading for both will be a step function for every circular path. A PI controller, illustrated in Figure 9, was implemented to maintain a change in heading by adjusting the steering of the agent when given a desired rate of heading change.

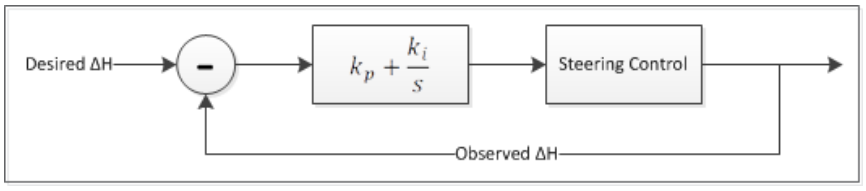

Figure 9: Block diagram of PID control of the heading's rate of change

Once the heading controller was tuned to provide a consistent heading change, a tracking algorithm was designed to convert the heading data into a displacement vector. While ΔH is constant, the paths created by the agent over uniform surfaces will always be circles. Instead of tracking the exact path of each arc, our tracking algorithm approximates these arcs with short chords. The length of a chord between any two points on a circle is given by the equation $L = 2rsin(c/2)$ where r is the radius of the circle, and c is the angle formed by the points and the center of the circle. The length of the actual distance traveled along the

arc is described by the equation $d = r * \Delta H/2pi$. When d is held constant over a specific time interval, these equations reduce to give chord length as a function of the H and ΔH. This chord length is the magnitude of the displacement vector, and the direction is equal to the average heading over the time interval in question.

With constant heading control and tracking out of the way, implementation of the Flight Plan Mobility Model is extremely straight forward. The ΔH controller was placed inside of a while loop that allows the controller to continuously feedback error signals and adjust heading as needed, and then an array was passed to a for loop surrounding this function. The for loop reads an array of flight path data, including the desired heading changes and length of time to maintain them, and then passes them to the ΔH controller.

The Smooth Turn Mobility Model facilitates on-the-fly generation of flight paths by way of exponentially distributed random arc distances and centers of rotation. Our implementation, shown in Figure 10, uses the exponentially distributed random number generator we designed in the Random Direction Mobility Model to create appropriate coast times before and after the center of rotation is relocated.

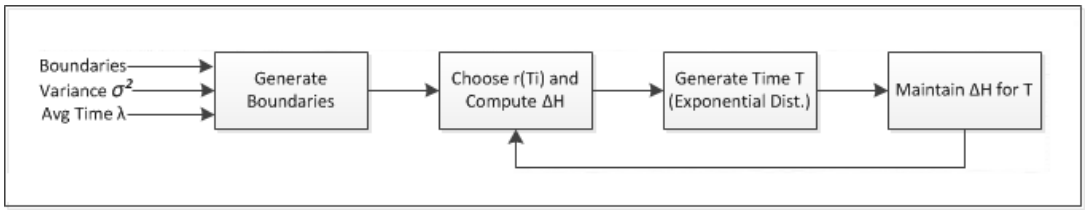

Figure 10: Block diagram of Smooth Turn Mobility Model

D. Decentralized Coordination

Like the Smooth Turn Mobility Model, the Flexible Stochastic Automaton-Based Algorithm for Network Self-Partitioning enables decisions to be made locally and on-the-fly at the agent level. This feature creates an added layer of security while simplifying the overall architecture of the system (See Figure 11). Our system partitions a network based on a search-and-rescue paradigm wherein agents who have discovered targets announce target acquisition to their nearest neighbors. This information spreads throughout the network by influencing the probability of other agents to change states to match that of the most influential local nodes.

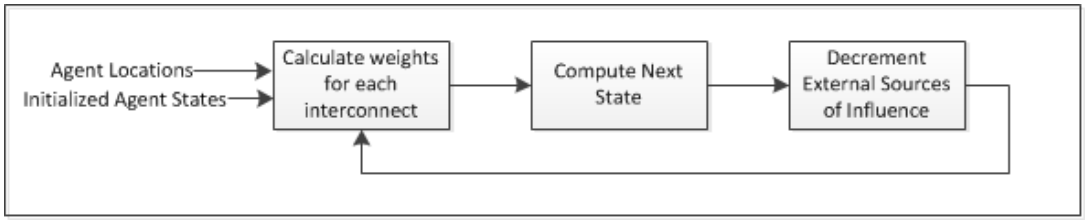

Figure 11: Block diagram Describing Decentralized Partitioning at the Agent Level

The movement of influence throughout the network is governed by the weights of the network interconnects. In a dynamic environment with constantly moving agents, these weights are constantly changing to reflect the agents' geographical distribution. This constant change forces the agents to reassess their neighbors for every iteration of the influence model. Because our robots are all more or less identical, the key contributing factor to the weight of a network interconnect is the distance between the interconnected nodes. A farther distance is obviously worse for communication and rendezvous, so the weights must have an inverse relationship with distance. In our example, the distance was subtracted from the maximum possible distance between robots. To emulate real-world wireless communication, a distance threshold was implemented to disable any interconnections that spanned too far. Figure 12 illustrates a network graph with

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

weights corresponding to geographical locations. In this example, the interconnect with a weight of 7.68 fell under the threshold of 8.0; this communication channel will be ignored by both Alpha and Beta.

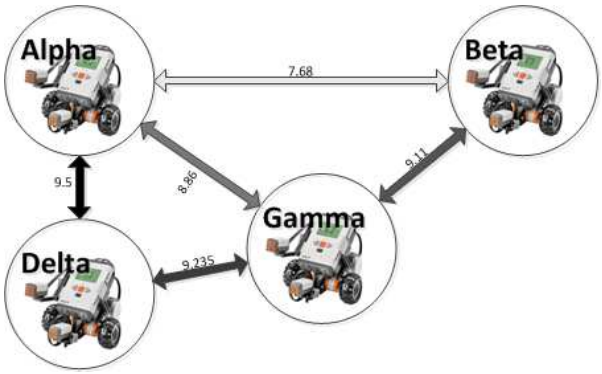

Figure 12: Example network snapshot with weighted branches.

Once the values of the interconnects are calculated, the algorithm converts them into probabilities. The agents begin the process with a low likelihood of keeping their state, and a subsequently high likelihood of being influenced by nearby nodes. This influence is directly based on the interconnection weights, so nodes with better connectedness have a better chance of spreading their state.

Initially, there may or may not be poor interconnects, but the ultimate goal of the algorithm is to produce them. This is accomplished by gradually weakening the likelihood that mismatched nodes will influence each other (and consequently strengthening the link between like nodes). This weakening continues until the interconnect weight falls below the threshold and is severed completely, partitioning the network.

E. Search and Rescue Application

A search and rescue implementation is comprised of four main stages: Initialization, Exploration, Partitioning and Target Aquisition. Each of these stages can be realized with the controllers and SubVIs outlined in this paper. To enhance this process, four discrete SubVIs were created to implement these main stages (See figure 13). The information supplied to the system consists of the agent names, their initial position, and the characteristics of their environment. The size, target count and branch threshold are pre-shared with all agents to facilitate exploration and the initialization of the partitioning algorithm. These arrays are sent to the initialization SubVI.

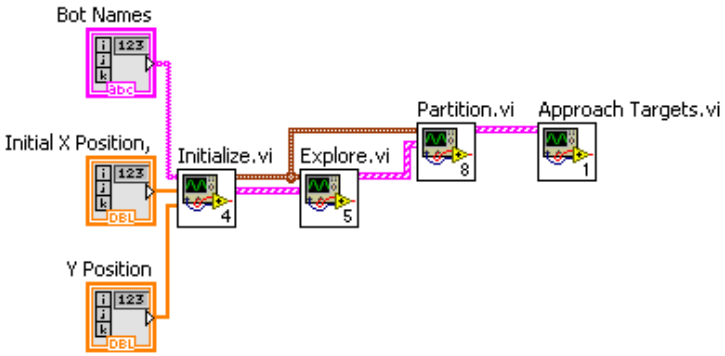

Figure 13: Overall block diagram of the search and rescue implementation.

The initialization SubVI generates an array of "Bot Clusters", clusters of data that represent the extents

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

of each agent's knowledge. Localization data, status and the NXT variable all reside within this cluster. This information is generated from the originally supplied agent names by initializing bluetooth communication with each agent. The data is easily accessible through the "unbundle by name" function, which leads to block diagrams that are much easier to understand, as illustrated in Figure 14.

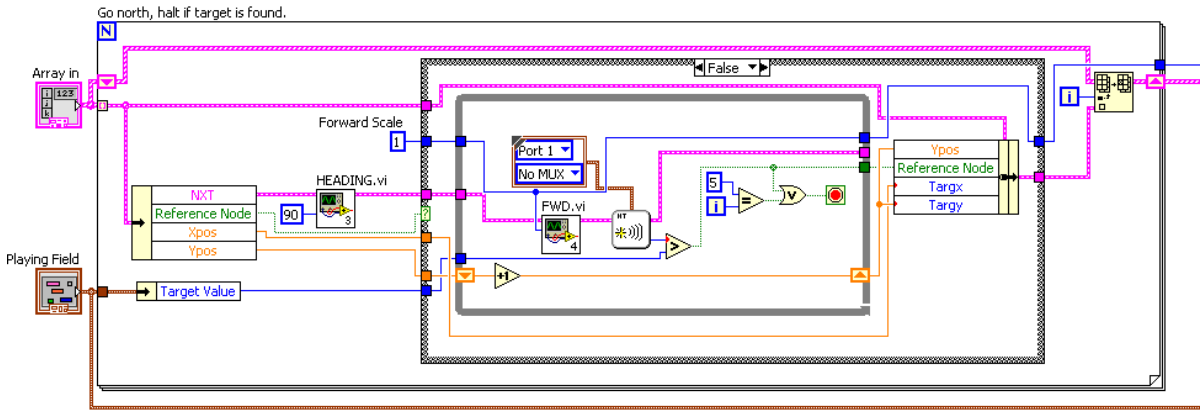

Figure 14: An example of unbundling-by-name indicating the read and written variables.

Exploration algorithms are asserted by creating a for loop that is capable of parallel execution so that each agent is commanded to move simultaneously. Agents will be instructed to explore the area until a target is found. Once all of the targets are discovered, partitioning begins. In the search and rescue demonstration, the statuses of each agent are initially null values. Agents that discover targets become reference nodes, their states become their location, and the remaining agents are influenced by these nodes. Because the resulting statuses are target locations, the final stage of the search and rescue operation is fairly trivial. A new planned trip is initiated with a final destination that matches that of the target.

IV. Results and Analysis

Overcoming the limitations of the Lego NXT proved to be consistently challenging. The platform is appealing due to its low cost and broad potential, but it is lacking many of the creature comforts found in most robotics systems. The most common challenge during implementation was the sample speed and execution time of sensors and actuators. In many cases, operation speed had to be sacrificed in order to provide an appreciable level of accuracy.

A. Basic Movement and Tracking

Nowhere was this more apparent than in the heading control portion of our work. Out of the box, an NXT robot rotating in place at a predefined power level for a specific amount of tire rotation (as measured by the built-in spin sensors inside the motors) could predictably end up within a range of about 30°. As illustrated in Figure 15, this range grew to approximately 75° at larger heading changes. Our rotational heading adjustment Sub-VI takes about twice as long to complete, but the resulting heading is accurate to within ±5° (Fig 16).

B. Random Exploration

The ground-based vehicle mobility models responded well to the accuracy of the heading Sub VI. The Random Direction Mobility Model draws attention to the strength of the heading adjustment algorithm at the boundaries of the playing field where the agent reflected off of a wall that existed only in software. Comparing the RD and RWP models in Figures 18 and 17 also draws attention to the difference between

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

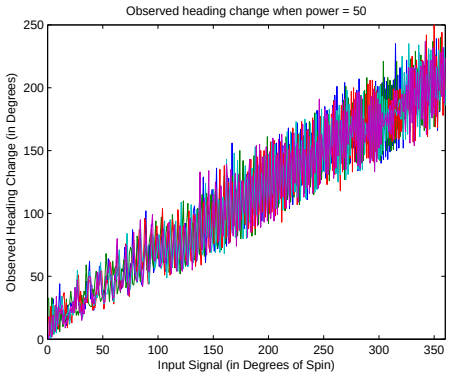

Figure 15: Heading adjustment using basic NXT blocks

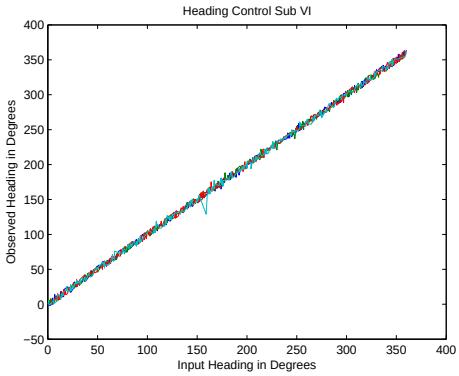

Figure 16: Heading adjustment using our closed-loop P controller

the two models. The corners of the Random Way-Point model are uniformly distributed, but the lines are clearly more concentrated towards the center of the field. The Random Direction model does not concentrate around the center of the map at all.

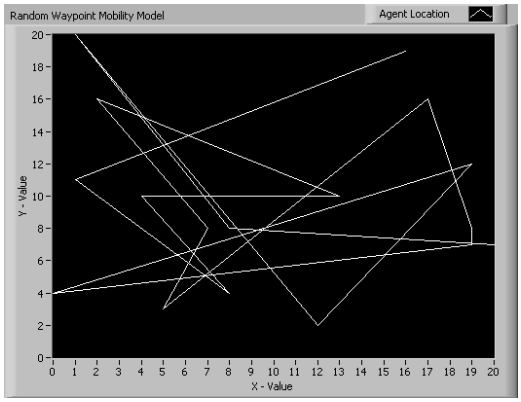

Figure 17: Tracking results of RWP Implementation

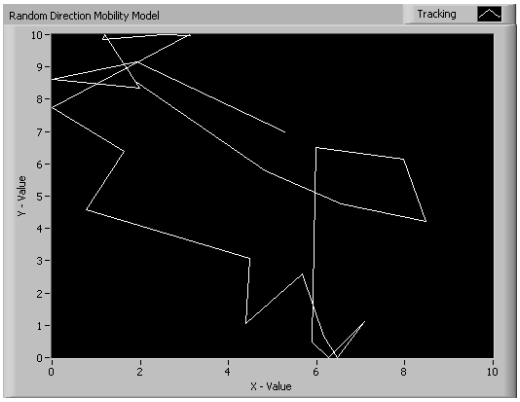

Figure 18: Tracking results of RD Implementation

C. Advanced Movement and Tracking

Airborne vehicle inspired mobility models demanded much greater control over the movement of the agents. Changing the heading while moving, in turn, demanded much more from the hardware: completing a single loop of the closed-loop ΔH controller Sub-VI takes almost 500 milliseconds. As a result of this extremely slow sample rate, the controller outputs a great deal of noise while tracking the input signal; however, this did not have a very large impact on the output of the Flight Plan and Smooth Turn Mobility Models, because integration helps attenuate high frequency noise.

The tracking data for the Flight Plan and Smooth Turn Mobility Models can be seen in Figures 19 and 20. The flight plan demonstrated was a simple spiral pattern beginning on the outside at $\Delta H = 0$ and gradually incrementing the ΔH controller Sub-VI, but any planned path is possible provided that it can be expressed in a series of headings and times. The smooth curves of the Smooth Turn Mobility Model illustrate that approximating an arc with a chord is not necessarily as jagged as it sounds; furthermore, neither model betrays the noisy nature of the ΔH controller due to the inherent low-pass filtering effect of integration.

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

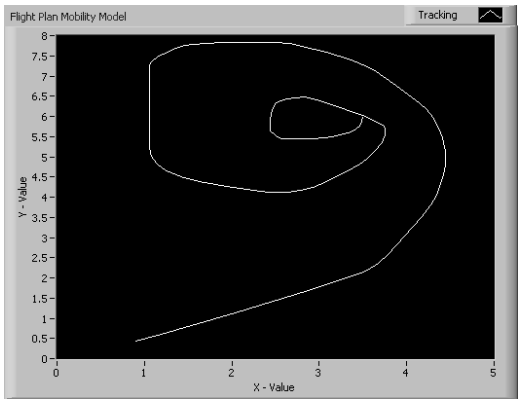

Figure 19: Tracking results of Flight Plan Implementation

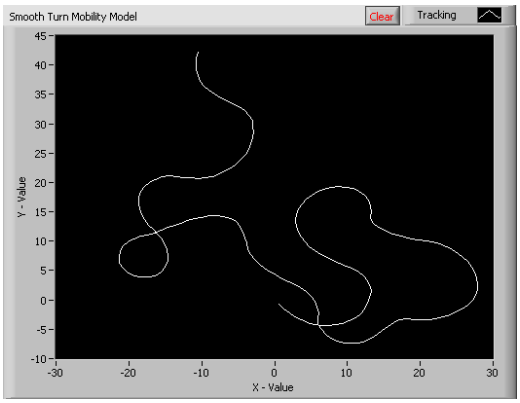

Figure 20: Tracking results of Smooth Turn Mobility Model Implementation

D. Decentralized Coordination

An example was implemented that demonstrated this algorithm on a group of five agents with a clear weak link in the middle that would be an ideal candidate for severing. In the example, each agent was initialized with a unique state before the partitioning algorithm began. As illustrated in Figure 21, agent statuses converge to a steady state, and the weight of the branch between the two distinct groups is reduced to zero. Reference nodes make this process more robust, and ensure that the network will reach the optimal solution with greater likelihood.

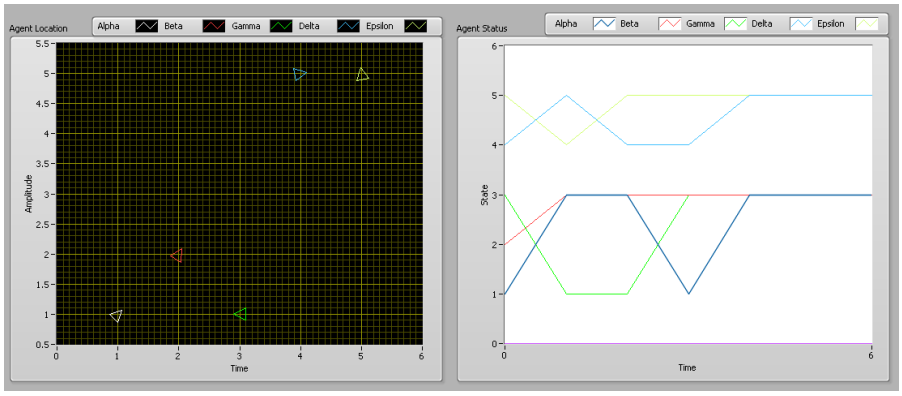

Figure 21: Results of the partitioning algorithm executed on a group of 5 agents

V. Conclusions and Discussion

The LEGO Mindstorms NXT Robotics Kit is a capable platform for Mobility and Coordination implementation. In this article we have implemented advanced mobility models that allow agents to maximize area coverage while keeping track of their position and network availability. More specifically, we have contributed the following to the study and implementation of advanced control system concepts:

1) Random Direction and Random Way-Point land-based mobility models were facilitated with a movement and tracking system that took advantage of the true rotation and true translation qualities of differential drive robots.

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

2) Airborne vehicle inspired mobility models that emphasized the requirements of gliding unmanned aerial vehicles were implemented with a more complex control and tracking system.

3) We also discussed the implementation of a flexible stochastic automaton-based algorithm for network self-partitioning to enable a decentralized robot swarm to make optimum search-and-rescue choices based solely on local data and a realistically constrained network architecture.

Acknowledgment

The first author is supported by the Research Initiation Grant from the University of North Texas. We would like to thank Dr. Sandip Roy at Washington State University for recommending the Lego Mindstorms NXT programmable robotics kits, and Mr. Yves Edimo Doualla for various discussions.

Downloaded by UNIVERSITY OF NORTH TEXAS on September 22, 2012 | http://arc.aiaa.org | DOI: 10.2514/6.2012-2568

References

[1]C. Asavathiratham, S. Roy, B. C. Lesieutre, and G. C. Verghese, "The influence model," *IEEE Control Systems Magazine*, 2001.

[2]C. Bettstetter, "On the connectivity of ad hoc networks," *The Computer Journal*, vol. 47, no. 4, 2004.

[3]C. Bettstetter, H. Hartenstein, and X. Peréz-costa, "Stochastic properties of the random waypoint mobility model," *Wirelss Networks*, vol. 10, pp. 555–567, 2004.

[4]S. Brigandi, J. Field, and Y. Wang, "A lego mindstorms nxt based multirobot system," in *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on*, july 2010, pp. 135 –139.

[5]Z. Cheng and W. B. Heinzelman, "Exploring long lifetime routing (llr) in ad hoc networks," in *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, 2004.

[6]R. Ghanta and S. Suresh, "Influence of mobility models on the performance of routing protocols in ad-hoc wireless networks," *IEEE 59th Vehicular Technology Conference*, pp. 2185–2189, 2004.

[7]W. Grega and A. Pilat, "Real-time control teaching using lego mindstorms nxt robot," in *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, oct. 2008, pp. 625 –628.

[8]Y. Kim, "Control systems lab using a lego mindstorms nxt motor system," in *Control Automation (MED), 2010 18th Mediterranean Conference on*, june 2010, pp. 173 –178.

[9]N. A. Lynch, *Distributed Algorithms*. San Mateo, CA: Springer, 1996.

[10]P. Nain, D. Towsley, B. Liu, and Z. Liu, "Properties of random direction," in *IEEE INFOCOM*, March 2005, pp. 1897–1907.

[11]A. Papoulis and S. U. Pillai, *Probability, random variables and stochastic processes*. McGraw-Hill, 2002.

[12]A. Tiwari, A. Ganguli, A. Sampath, D. Anderson, B.-h. Shen, N. Krishnamurthi, J. Yadegar, M. Gerla, and D. Krzysiak, "Mobility aware routing for the airborne network backbone," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, nov. 2008, pp. 1 –7.

[13]Y. Wan, D. He, and K. Namuduri, "A smooth-turn mobility model for airborne networks," 2011, unpublished abstract, to be submitted.

[14]Y. Wan, S. Roy, A. Saberi, and B. Lesieutre, "A flexible stochastic automaton-based algorithm for network self-partitioning," *Int. J. Distrib. Sen. Netw.*, vol. 4, pp. 223–246, July 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1451833.1451834

[15]F. Xue and P. R. Kumar, "The number of neighbors needed for connectivity of wireless networks," *The Computer Journal*, vol. 10, pp. 169–181, 2004.