A LEGO MINDSTORMS NXT BASED TEST BENCH FOR MULTIAGENT EXPLORATORY SYSTEMS AND DISTRIBUTED NETWORK PARTITIONING

Riya Raghuvir Patil, B.E

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

May 2014

APPROVED:

Yan Wan, Major Professor
Kamesh Namaduri, Committee Member
Shengli Fu, Committee Member and Chair of the Department of Electrical Engineering
Costas Tsatoulis, Dean of the College of Engineering
Mark Wardell, Dean of the Toulouse Graduate School

Patil, Riya Raghuvir. A Lego Mindstorms NXT Based Test Bench for Multiagent Exploratory Systems and Distributed Network Partitioning. Master of Science (Electrical Engineering), May 2014, 47 pp., 25 figures, references, 31titles.

Networks of communicating agents require distributed algorithms for a variety of tasks in the field of network analysis and control. For applications such as swarms of autonomous vehicles, ad hoc and wireless sensor networks, and such military and civilian applications as exploring and patrolling a robust autonomous system that uses a distributed algorithm for self-partitioning can be significantly helpful. A single team of autonomous vehicles in a field may need to self-dissemble into multiple teams, conducive to completing multiple control tasks. Moreover, because communicating agents are subject to changes, namely, addition or failure of an agent or link, a distributed or decentralized algorithm is favorable over having a central agent. A framework to help with the study of self-partitioning of such multi agent systems that have most basic mobility model not only saves our time in conception but also gives us a cost effective prototype without negotiating the physical realization of the proposed idea.

In this thesis I present my work on the implementation of a flexible and distributed stochastic partitioning algorithm on the Lego® Mindstorms' NXT on a graphical programming platform using National Instruments' LabVIEW™ forming a team of communicating agents via NXT-Bee radio module. We single out mobility, communication and self-partition as the core elements of the work. The goal is to randomly explore a precinct for reference sites. Agents who have discovered the reference sites announce their target acquisition to form a network formed based upon the distance of each agent with the other wherein the self-partitioning begins to find an optimal partition. Further, to illustrate the work, an experimental test-bench of five Lego NXT robots is presented.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude towards Dr. Yan Wan for her unwavering support, excellent guidance, unrelenting patience, and for providing me with a professional atmosphere for doing research. I would like to thank Dr. Shengli Fu and Dr. Kamesh Namaduri for supporting my research and being willing to participate in my defense committee. I would also like to thank Dr. Yan Wan and Dr. Murali Varanasi for supporting me financially throughout my M.S. degree.

I am particularly grateful to Vardhman Sheth, who as a good friend was always willing to help and give his best suggestions. Without his persistent help and guidance, this dissertation would not have been possible. I would like to thank the Graduate Writing Support Center, especially Noah Geisert, for helping me with put up my ideas clearly in the writing. Many thanks to my lab-mates Junfie Xie, Vardhman Sheth, Yi Zhou, and Gopichand Movva for your insightful observations. Also thanks to my friends at the University Of North Texas: Nikhil Shah, Vidusha Kanth, Pranav Tripathi, Rahul Angural, and Sherin Abraham.

Special thanks go to my brother Raj Patil for his immense support and encouragement throughout my journey. I would like to thank my family Mr. Raghuvir Patil and Mrs. Anjali Patil, Mr. Sanjay Patil and Mrs. Sujata Patil, and Mr. Vinay Raut and Mrs.Seema Raut for supporting me spiritually throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 Motivation and Background

Animals that travel in groups exhibit a variety of mobility patterns through fascinating detailed interactions. A familiar but remarkable example includes the flocking of birds, their abrupt splitting, their unity in flying, their sudden change of course, and their synchronized landing, all scenarios that demonstrate rapid co-operative decision-making. In this thesis, I take inspiration from this avian behavior of such an inter-individual interaction, where individuals both influence the other group members into a collective behavior, yet seemingly carry out an action that is leaderless, decentralized, thus giving us insights for similar models of self-organization. It suggests that much of the complex group behavior can be coordinated by relatively simple interaction of the group.

There are ample embedded computational means in autonomous vehicles that enable improved operational potency through cooperative teamwork. Multi vehicle systems are an important category of networked systems due to their commercial and military applications [28] and that offer considerable advantages over an individual agent. In comparison to autonomous entities that perform solo missions, greater efficiency and operational ability can be comprehended from the teams of autonomous vehicles operating in coordinated mannerisms. Potential applications for multivehicle systems include space based interferometers, surveillance systems, combat, and reconnaissance systems as in military and civil operations, planetary exploration distributed re- configurable sensor networks. To back these applications a need arises to develop various co-operative control capabilities: formation control flocking, foraging, , exploration, rendezvous attitude alignment, task and role

assignment, air traffic control and cooperative search, search and destroy, and search and rescue algorithms.

Distributed algorithms for self-partitioning may be valuable for various sensor networking and autonomous vehicle control applications. Need arises when a swarm of autonomous vehicles in a field have got to self-organize themselves in order to accomplish multiple control tasks all at the same time. To infuse a real-time behavior, with a variety of situations, considering their environment and the treatment of unexpected events can be quite challenging. It is necessary to simulate and analyze the main functions, design and analysis of each agent in an autonomous and decentralized multi agent robotic system. Also it is important to find out an efficient strategy to realize effective means for the agents to form a cooperative manner that supports target exploration and self-partitioning. In the latter section, I provide some pointers to the literature on related systems and briefly note their difference with my test model.

A motivation for the model, where it can be applied, is in a planetary mission. Robotic planetary missions cover a wide range of scenarios from grabbing a piece of rock near the space craft landing site to building and maintaining a large radio observatory on the lunar front. I discuss how a team of autonomous robots can be applied to similar missions. A team of small autonomous robots can provide savings in launch mass, landing mass, and fabrication costs [5].

1.2 Contribution to the Research

Various multi-robot systems have been developed in the last decades. There has been an increased interest in the field of distributed robotic systems due to its promising applications in many fields. These systems are desirable in realizing an advanced robot system to achieve

arch level tasks. This thesis is a contribution to multi-agent systems that interact with its neighbors to self-partition themselves.

To build such interacting multi-agent models, I emphasize on its physical construction, communication and algorithms. When realizing the (a) physical construction I focus on actual robotic units that form the systems. I concentrate on their mobile behavior such that they exhibit a realistic behavior. I take care of (b) communication, such that it is robust and reliable, and try to realize its effective means for units to form a cooperating and interacting system. Last, for the core of the model I give prominence to the (c) algorithms used to create an effective realistic model [9].

A starting point to this research is to examine the applicability of the existing systems for autonomous and decentralized co-operative robots. Research on robotics has primarily focused on enabling self-capable robots and a number of algorithms that utilize these robots to accomplish a common task [13]. Approaches have been developed towards enabling cooperation to instill wireless communication and networking capabilities on the robots. The robot teams are built to undertake search and rescue missions [3] [7] [15]. [9] discusses simulation environment for autonomous and multi agent decentralized robotic system, an intelligent robot system that is capable of executing high level tasks.

Various studies for multi-robot systems have been developed that dealt with algorithms focusing on a construction of system that could search an environment for randomly placed objects and push them to a predetermined space [16]. The study involved in the development of an algorithm exercised explorative searching and dynamic task allocation. Multi-agent rendezvous problems have also been discussed in [14]. Distributed robotic systems that portray

a self-organizing behavior and swarm intelligence using mobile robots have been addressed in [12] [13].

My work revolves around the partitioning algorithm [1]. I take a note of the consensus algorithms used to impose similar dynamics on the information states of each vehicle, with the communication network among the vehicles allowing continuous information updates. [10]. Collective group behavior through local interaction is also explained in [10] [20] [21] [22]. Shared information is necessary for co-ordination. One needs to know what information and to whom the information needs to be shared [11]. I use LEGO® MINDSTORMS® NXT devices as my mobile devices to form a multi robot system. Works have been published that have used the NXT devices as multi-agent systems [8].

1.3 Objectives of the Research

The consensus algorithms are designed to be distributed, assuming neighbor to neighbor interaction between vehicles. Vehicles update the value of their information state based on the information states of their neighbors. The goal is to design a standalone unit such that the information states of all the vehicles converge to a steady state, with each vehicle deciding what state is suitable for itself. In other words, I have developed an automaton that can group vehicles in such a manner that the self-assembly takes a little time, and the vehicles/robots/agents in the group can easily communicate.

I provide a test bed of a group of mobile robots that have mostly the same physical structure and self-decision making capability to track targets, to form a network and to self-partition depending upon the influence of its distances (that correspond to probabilities) with the other robots. The partitioning is decided by each robot with the help of the information acquired by each of its neighbors from its corresponding immediate neighbors. The

information is mainly about changes in the local network influences and is retrieved at every time-step using simple mathematical calculations. Each of the agents integrates a basic collection of software components, including a perception system, or a tracking system, such as a light intensity sensor, a motion system, and a motion controlled strategy defined as a mobility model, a communication module for agent to agent interaction which enables signaling or passing the status information to the neighbors.

I infused in the robot a mobility model and an odometric system to explore targets and acquired them. I then classified a swarm of robots using a classifying or partitioning algorithm called as copying influence model [1] [2] for self-classification into two distinct statuses which are nothing but distinct target locations.

1.4 Organization of the Thesis

The thesis report is organized as follows: Chapter 1: Introduction. Chapter 2 presents the fundamentals that form the basis of this research. It explains several hardware features such as LEGO® MINDSTORMS® NXT and, NXTBee radio module that are used in this test bench. Chapter 3 describes mobility models and gives reasons for the use of the mobility model I have implemented particularly. It makes familiar the communication strategies that can be used, and describes the communication platform I have implemented. Chapter 4 brings forward the distributed partitioning algorithm, the behavior of individual agents to and the multi agent co-ordination developed in the test-bed. Certain examples are used in all the sections to illustrate various points. Finally, I have developed a demonstration to put forth a unit that justifies the title of this thesis, i.e. “A LEGO® MINDSTORMS® NXT based test-bench for Multi-agent Exploratory System and Distributed Network Partitioning.” The last chapter,

Chapter 5, analyzes and gives appreciable results and concludes the thesis with the potential to work further on developing the automated unit.

CHAPTER 2

MULTI ROBOT SYSTEMS

2.1 Introduction

Practical implementations reflecting real time instances can be quite challenging. One has to choose an appropriate physical system meeting several academic as well as practical needs. Some of the essentials of a system are to 1) being able to replicate the experiment like a real time environment to a substantial extent, 2) work within an existing software environment, 3) be low cost 4) be locally purchasable for easy maintenance, and 5) be easily programmable and reconfigurable.

The LEGO® MINDSTORMS® NXT provides a viable solution to accommodate all the above needs. The NXT is a commercial kit that the Lego Company sells for building various types of robots. The NXT's interesting attribute is its mini-computer called NXT-brick and its various other LEGO components including sensors and motors. The motor system has enough inertia for educational purposes with the result that the combination of appropriate software can serve as an ideal system for various test benches.

Similarly, the software that interfaces with the hardware must: 1) support the hardware justifying its use to a full, appreciable extent, 2) generate an executable program that can directly be and compiled and uploaded on the hardware, 3) run independently of the computer, and 4) provide powerful and varied functionality.

National Instruments' LabVIEW is used as a software interface with the LEGO® MINDSTORMS® NXT to extract the best out of the system. This section gives us an overview of the software and hardware platforms. Also discussed later in the section, is the wireless communication strategy used in the test bed.

2.2 Hardware Platform: LEGO® MINDSTORMS® NXT

The use of LEGO® MINDSTORMS® NXT robots has become quite popular in recent years in implementing classical and modern control theories. When combining the NXT devices with a programming platform, it becomes possible to showcase frameworks related to modeling, state feedback control, estimator designing, and so forth. The Lego building systems has enabled engineers to build, to program, and to have hands on research in real-life robotic solutions.

I chose LEGO® MINDSTORMS® NXT to be the agent prototype due to its low cost but diverse functionality and flexible re-configurability. The LEGO® MINDSTORMS® NXT robotics kit costs from $294.95. It comes with a programmable NXT Brick, a 32-bit arm processor providing on-brick programming and data logging; interactive servo motors; ultrasonic, sound, light, and touch sensors; a rechargeable battery; and connecting cables. The aim is to allow a team of robots to perform a combination of explorative searching and dynamic task allocations.

Figure 1: LEGO® MINDSTORMS® NXT: NXT Brick, NXT-Robot, light intensity sensor.

A NXT robot can be individually programmed or can be connected to a computer via a USB cable. The test bench consists of a team of five robots programmed robots to be individuals and is a unit of five standalones. The unit runs independently, and without a computer since I

pre-program each robots. I chose the most basic three-wheeled design for my agents: two wheels, each operating independently, and a rear wheel for stability. All of the agents are more or less functionally and structurally the same. Thus, each of the agents is an NXT robot built with two motors (rear wheels) for mobility, a light sensor for exploratory properties, and an NXTBee sensor for communication. Thus conformed all five robots identically with the above sensory, mobility, and communication features. Such five robots together, called as agents, formed a multi-robot or multi-agent automaton. A prototype of one of the agent robot is shown in the figure below:

Figure 2: An agent finds a reference site

2.3 Software Platform: LabVIEW™

Numerous programming languages can be used to program the NXT including National Instruments' LabVIEW™, NXT-G, RobotC, Next byte Codes and Not eXactly C. I use the LabVIEW™ environment to program the agents as LabVIEW™ utilizes NXT and its sensors to their full extent, and LabVIEW™ also allows data logging.

National Instruments' LabVIEW™ is a graphical programming tool ideal for any measurement or control system. It is useful in a broad range of applications and is relatively easy and time saving for multiple operating systems and devices [23]. The block diagram below gives an idea of a single graphical development platform.

Figure 3: LabVIEW™ is a single graphical programming platform.

The LabVIEW™ for LEGO® MINDSTORMS® NXT is designed specifically for the use of the Lego Education robotics platform. While one can graphically configure and test motor and sensor connections, one can also easily log and analyze that data that is collected from NXT.

2.4 Wireless Communication

For the agents to communicate with each other it is extremely crucial to have a smooth and a robust communication. As explained earlier, LabVIEW™ is a command box programming tool so the time required to execute a block in LabVIEW™ is relatively more than the time required in executing a few lines (those analogous to the block) in command line programming tool (E.g. RobotC). This particularly makes the communication between two NXT devices even

more challenging to make it smooth, perfect and as fast as possible. Moreover, adding delays would slow the automaton, but would ensure a trustworthy connection. Nevertheless, in attempting to achieve a communication as smooth as possible, I have kept in mind that the automaton should not be a unit slow to the extreme, but should be appreciably fast. This section explains the cons of using the built-in Bluetooth technology of the NXT device and my reasoning for why I have used the XBee radio module instead of using the built-in Bluetooth.

2.4.1 Bluetooth Wireless Technology

Bluetooth is a wireless technology of the IEEE 802.15.1 standard. It operates at 2.4 GHz frequency and can exchange data over a physical range of up to 100 meters. The NXT devices are built-in with the Bluetooth technology. This Bluetooth feature can be used to connect the NXT devices to other NXT units, mobile phones, and other computers. Under ideal conditions such as flat ground, no wireless interference, no walls or obstacles between NXT robots, Bluetooth can communicate up to 15-20 feet (4.5-6 meters). Although Bluetooth communication is robust and reliable, the number of devices that can be connected together are limited. A group of connected robots can contain only up to four NXT devices (Lego Bluetooth manual). Moreover, Bluetooth in NXT devices uses master-slave model in which the unidirectional control of the master NXT is limited to only three devices. Even supposing that I switch the NXT agents as a master or slave; it is unconvincing to make communication so complex.

So, I shifted my effort to a radio module called XBee sensor that is capable of broadcasting information to the NXT agents under similar ideal conditions. The XBee radio system is a versatile communication solution for the LEGO® MINDSTORMS® NXT system. These high-speed long-distance radios allow your NXT to communicate with any other device

with an XBee radio. The performance of the NXTBee greatly exceeds Bluetooth, infrared, and other forms of communication currently available for the NXT.

2.4.2 XBee Wireless Radio Technology

XBee is a product of Digi International. It supports IEEE 802.15.4 networking protocol which is the standard for robots everywhere. It includes point-to-point networking and meshes networking. It operates at 2.4 GHz frequency and can be used to exchange data up to 300 feet (90 meters) and the XBee-pro up to 1.2 km; 10 times farther than Bluetooth or IR. The Dexter Industries NXTBee sensor brings the Digi XBee radios to the LEGO® MINDSTORMS® NXT. The Digi XBee module uses the Dexter Industries' adapter when interfacing with the NXT devices (NXT Bee). The NXTBee utilizes the NXT's high speed RS-485 line for high speed communication. The NXTBee's standout feature is its ability to mesh devices together i.e. to create a mesh network. NXTBee automatically finds other XBee radios (other NXTBees or devices using XBees) and can relay information to them and between them. A notable thing is that the NXTBee needs no set up as it is done automatically. XBee radios pass on data and also gather information from other XBees, ensuring a two-way communication. Dexter Industries claims that one can create a whole network of 65,000 different devices talking to each other. Each agent in my test bed is equipped with an NXTBee sensor to receive as well as to pass on information, thus aiding inter robot communication. I have discussed the usability and relevance of all the above components in the chapters ahead.

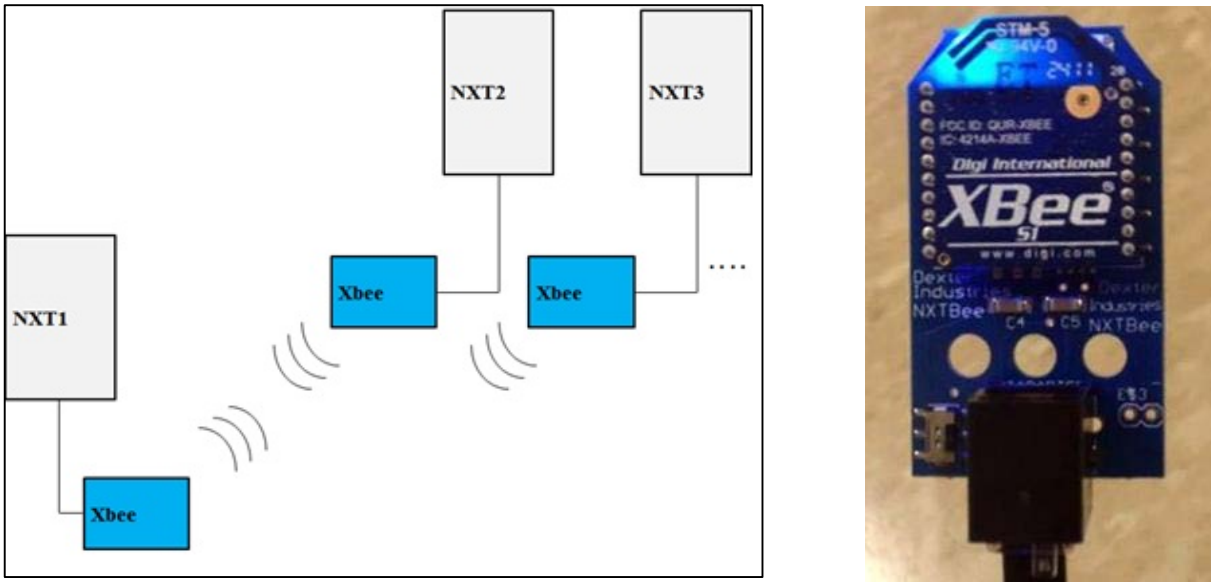

Figure 4: NXTBee communication point to multipoint communication

CHAPTER 3

RANDOM EXPLORATION

3.1 Introduction

My test bed's purpose is to have an interacting network model for distributed partitioning. The first step ensures to replicate real time movement in the NXT-robots like those of the unmanned vehicles or exploratory systems. I attempt to develop a model of exploratory search behavior; thus form a robust system of multi-agents each capable of exploration.

The following chapter gives a detailed knowledge about the random exploration using a random direction (RD) mobility model. One gets a brief idea about the mobility models and what led to the choice of the RD model. With the help of LabVIEW™ as a programming tool and LEGO NXT devices and sensors, the chapter explains the implementation and test set up and the communication strategies to form a network.

3.2 Mobility Models- Random Exploration

A mobility model is intended to describe the movement pattern of mobile users based on the location, velocity and acceleration change over time. Mobility models are commonly used in studies related to mobile and ad hoc networks. They form the premise for the evaluation of communication strategies and routing protocols [25]. The purpose of using a mobility model is to have agents perform uniformly distributed random exploration in a predefined area and search for targets.

3.2.1 Choosing a Mobility Model

A number of mobility models have been proposed for analysis or simulations of the movement of users in a mobile wireless network [4]. We need an abstracted mobility model that

simplifies the agents' act of exploration. A planned exploration or a human controlled model via tele-operation is encouraged, but what interests us is random exploration.

In random-based mobility models, the mobile nodes move randomly and freely without restrictions. To be more specific, the destination, speed and direction are all chosen randomly and independently of other nodes. Two of the ground based mobility models (also aerial based) are the random way-point model and the random direction model. Of these, the random direction mobility model, although physically unappealing, is easy to understand, and yet realistically reflects the real world scenario and its constraints. The simplicity of this model is that the positions and the directions are uniformly distributed [24]. The random direction mobility model captures realistic mobility patterns and can produce large trajectory ensembles [3]. Since, the main goal is to showcase an automaton for distributed partitioning; we keep to the use of basic random direction mobility model.

3.2.2 Random Direction Model (RD Model)

The random direction model or the RD model explores a predefined area by randomly choosing a direction and moves till the boundary is reached. An agent, thus, chooses a random direction (α) from a uniform distribution from $-\pi$ to π (with probability density function $f(\alpha) = \frac{1}{2\pi}$) and moves till the boundary reaches. It then chooses another random direction, α, till the boundary is met again [3]. My test bed uses a modified RD model; it explores an area by randomly choosing a direction and a distance and moving until the target is found. It takes a random direction, moves forward for an exponential period of time T, with probability density function $f(T) = me^{-m_d T}$ where $\frac{1}{m_d}$ is the mean duration. After the duration completes, the agent then chooses a new random direction. Exponential distribution is used for its memory less

property. Random direction mobility model can also be developed considering boundary reflection.

3.2.3 LabVIEW™ Implementation

The LabVIEW™ implementation takes care of the movement and tracking of agents in the test bench. The movement is the random direction mobility model in which the agent will choose a random heading, i.e., direction, and then move forward until the target is found. The random exploration is shown by the flow chart i.e. Figure 5.

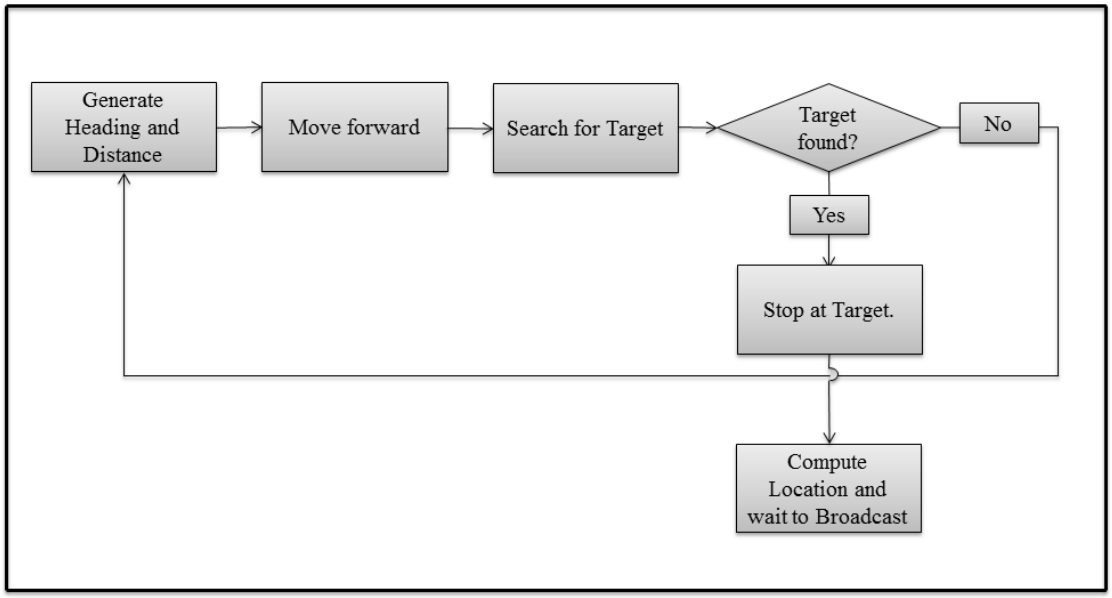

Figure 5: Overview of the random exploration

We divide the RD model as 1) heading 2) distance change, i.e., the forward movement of the agent. While the heading changes are uniformly distributed, the forward movement of the agent is distributed exponentially.

The agent's NXT motor has a built-in encoder with the resolution of 360 ticks per revolution. We use the built-in encoder to control and track the agent's heading and travelling

distance. Vardhman et. al. claim that the encoder shows better precision than the angle sensor and the compass sensor. If T was the time required for an agent to travel in the forward direction, the distance required to travel can be given by

$$distance = v \times T$$

……. (1)

where, T is randomly selected from the exponential distribution$f(T)$ and v is the speed of the agent. As the robot changes the heading, the changing angle $\propto$ is selected from the uniform distribution; both the wheels of the agent are allowed to move opposite to each other instead of letting just one wheel to move. This helps realize the heading change and also helps to reduce the rotation of the wheels.

The NXT motors do not have a high level of precision. So I developed a proportional controller.

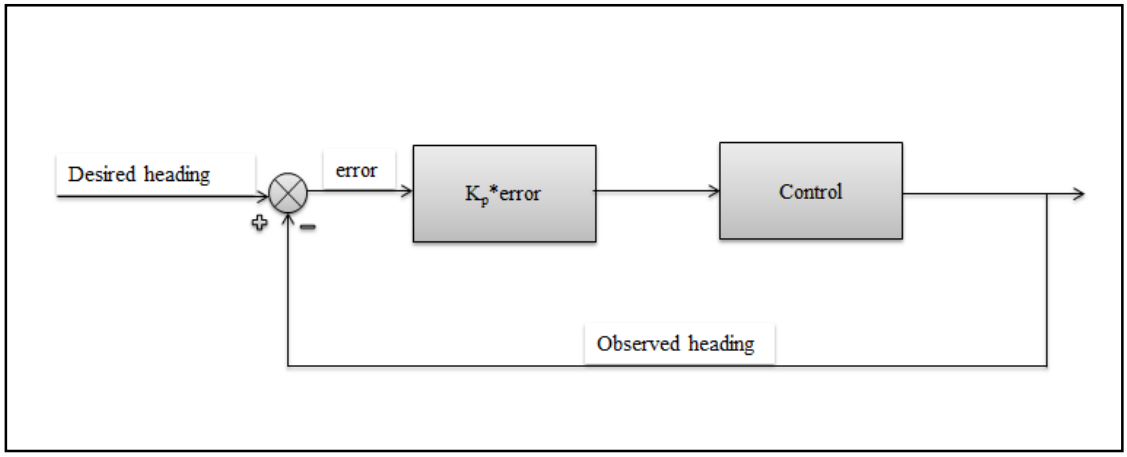

Figure 6: Proportional controller used for the RD model

The reference heading or desired heading $'\alpha'$ is compared to the encoder reading which can be calculated from the equation given below:

$$\propto = 2(e_c \, {}^{2\pi r_{wh}}/_{2\pi l})$$

….. (2)

$$distance = e_c \, {}^{2\pi r_{wh}}/_{e_{res}}$$

….. (3)

where, e_c is the count of the encoder of the motor and its resolution is given by e_{res} , while r_{wh} is the radius of the wheel. And the output (i.e., the power from the equation (5)) from the proportional block actuates the motor.

$$Err = \propto - current\ Reading$$

….. (4)

$$Power = Kp \times Err$$

….. (5)

The proportional controller permits better precision of the heading direction change. I developed the heading adjustment SubVI that rotates the agent to the desired heading.

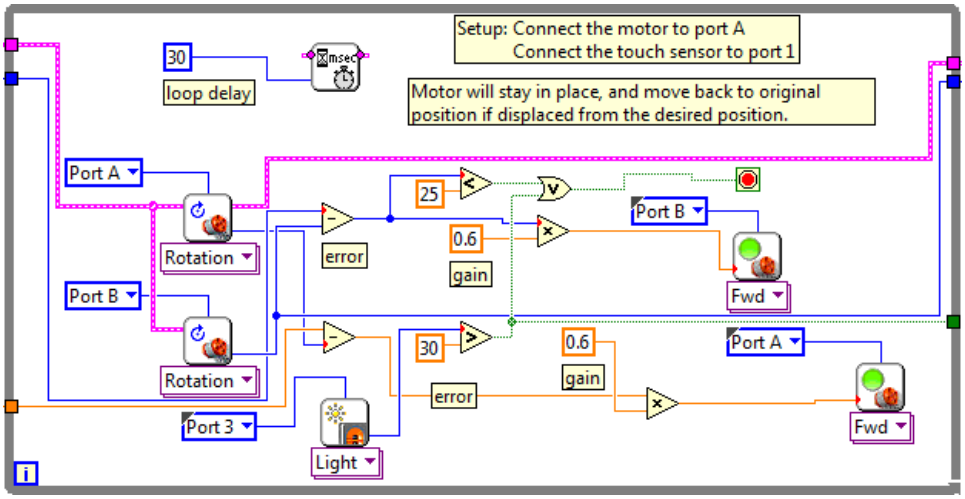

Figure 7: LabVIEW™ snippet for heading adjustment in the RD model

There is an important feature of LabVIEW™, where multi-thread is successfully used to carry out two or more operations simultaneously. Multithreading extends the idea of multitasking into applications, so that specific operations within a single application can be subdivided into individual threads, each of which can then run in parallel. Similarly, in a LabVIEW™ multithreaded program, the application can be divided into three threads – a user interface, a data acquisition, and a control– each of which can be assigned a priority and can operate independently. Thus, multithreaded applications can have multiple tasks progressing in parallel along with other applications. I have used this multi-threading capability of LabVIEW™ to simultaneously run the RD model, search for targets, and to receive information from the agents of target acquisition resulting in random exploration. Let us look at the test bench set up of the randomized exploration.

3.3 Test Bench Setup

Once the random direction model is established and implemented in each of the agents, we can focus on the exploration of the targets. To track the targets each agent is provided with a light intensity sensor. This section acquaints one with the arrangement of the test bed that accomplishes random exploration.

The test set up has five agents, each is an LEGO® MINDSTORMS® NXT equipped with a light intensity sensor and an NXTBee sensor in a pre-defined area. Figure 8 and Figure 9 show the test field and the initial positions of the five agents.

Figure 8: Test field

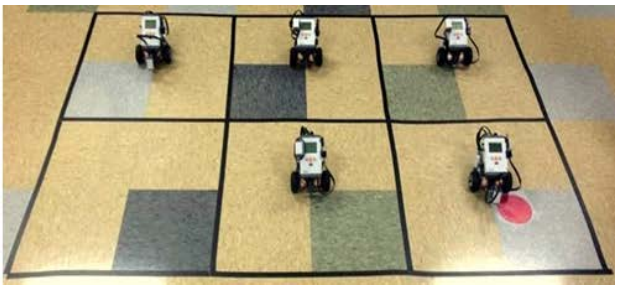

Figure 9: Initial position of the agents

The two red circles/spots denote the reference sites. All the five agents search the area with the help of the light intensity sensor. The exploration is complete once the two reference targets are found. The agents who hit the reference sites announce their acquisition. When all the reference sites are found, the agents stop in their current position. The current position is independent of the heading and the position of the agents. With all the agents at a halt, communication of the agents begins to form a network, which is supported with the NXTBee module to collect the information of the current location and statuses of all the agents. The information of the loctions af all the agents is required as we form a network, for distributed partitioning; that comprises of reference sites and the sites of the other agents.

The grid shown above is independent of the movement and tracking of the agents. The agents can explore the entire area. The agents are not bounded to the periphery of the grid. The grid is a referrel to the co-ordinate system carved for localization. The area and so the agents are in the positive quadrant of the axis, each of them oriented at 90 degrees to the axis, when in their initial position. The co-ordinate axis represents the *x-y* plane. The length of the rectangle represents the x-axis and the breadth, the *y*- axis. The alphabetical letters in *xy* plane represent the co-ordinates in the plane. The smallest of the blocks in the grid represents the alphabet inside

it and the whole block is considered as one co-ordinate. This compromise was done to aid the communication between the agents.

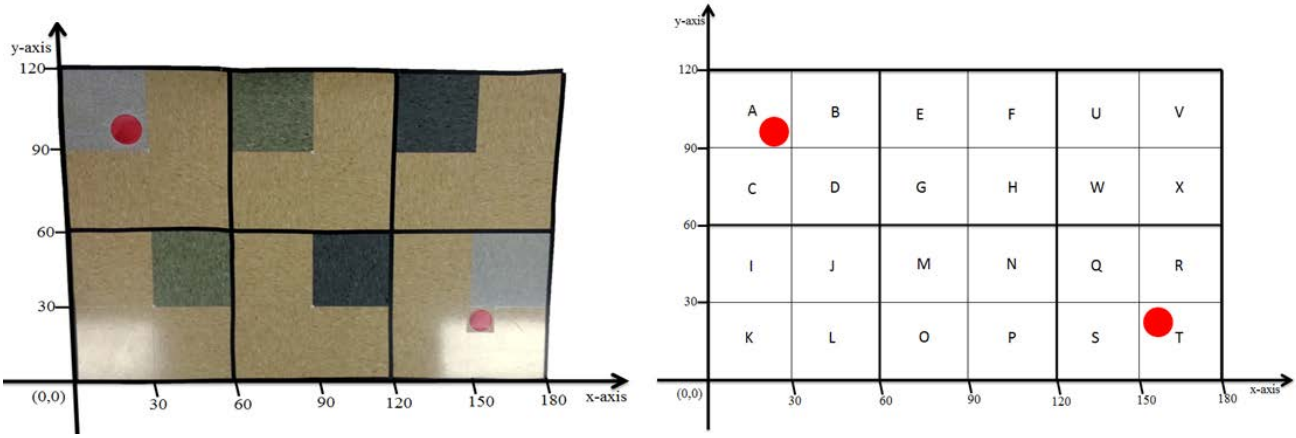

Figure 10: A snapshot of the test field and its simplified interpretation of the coordinate axis developed for the test-bed

Mathematically, a tracking algorithm was developed to track the location of the robots. If 't' was the index to change direction or the heading, τ as the angle to change at *t*, s_t as the distance to travel for the t^{th} directional movement then the following set of calculations help us to track the position of the agents.

The heading of the agent (w.r.t to the *x*-axis), can be calculated using equation (6). The value of is kept in between *-180* to *180*, where mod is the modular operator. The location of the robot can be calculated using equation (7), where *(x, y)* represents the coordinates of the robot

$$\theta_t = mod(\theta_{t-1} + \tau_{\ t} + 180{,}180) - 180 \quad \ldots\ldots (6)$$

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + s_t \begin{bmatrix} cos\ \theta_t \\ sin\ \theta_t \end{bmatrix} \quad \ldots\ldots (7)$$

3.4 Communication: Formation of the Network

The NXTBee sensor is used for the communication between the agents. The, NXTBee sensor helps in informing on the location of each agent's position, i.e ., its x and y co-ordinates, which needs to be broadcasted so that a network can be formed. The NXTBee sensor comes with its own limitation of its ability to send or receive single string at a time. The co-ordinates of the location, hence, were stashed in the series of alphabetized letters(as explained in the previous section). The agent would then broadcast an alphabet corresponding to its location. The rest of the agents match the received location from their cached table. This not only benefited in sending information of the location but also simplified the communication.

As soon as any agent would broadcast its location information the remaining agents would calculate their distance with this sending agent. The criteria to form a link or a connection between any two agents was a threshold distance. What I call the 'threshold distance' is a fixed maximum distance between any two agents that will still allow them to link. If the distance between the agents is less than the threshold distance, a connection is formed, or else it is not. This continues till all the agents broadcast their location and form a network.

The network that forms depends upon the acquisition of the targets and when the agents halt after the acquisiton announcement. Hence, the network can take any shape until the targets or reference sites are acquired. This checks about the stochasticity and the flexibility of the network. The next chapter deals with the further implementation of the flexible partitioning algorithm of the stochastic network.

CHAPTER 4

DISTRIBUTED PARTITIONING

4.1 Introduction

This chapter concentrates on the implementation of flexible and distributed stochastic automaton capable of finding an optimal *k*-way partitioning algorithm with respect to a broad range of cost functions, and given various constraints [1]. Given a set of hardware components, I have shown how a partitioning algorithm can be incorporated while considering the real-time constraints stated in the system's specification.

To establish the foundation for the distributed partitioning algorithm, the beginning of the chapter acquaints one with the general influence model, which is a self-partitioning model and a modified case of the influence model that I have implemented. The latter part of this chapter discusses about the implementation with the Lego NXT devices using the graphical LabVIEW™ interface. Finally, one gets to interpret the impressive achievement of the agents that self-organize and carry out the adaptive and iterative self-division.

4.1.1 Mathematical Background

Consider a graph of vertices or nodes or sites and weighted edges as shown in the examples given in Figure 11, in both the cases, the system is made of a finite number of states, and transitions between states occur at discrete instants according to specified probabilities.

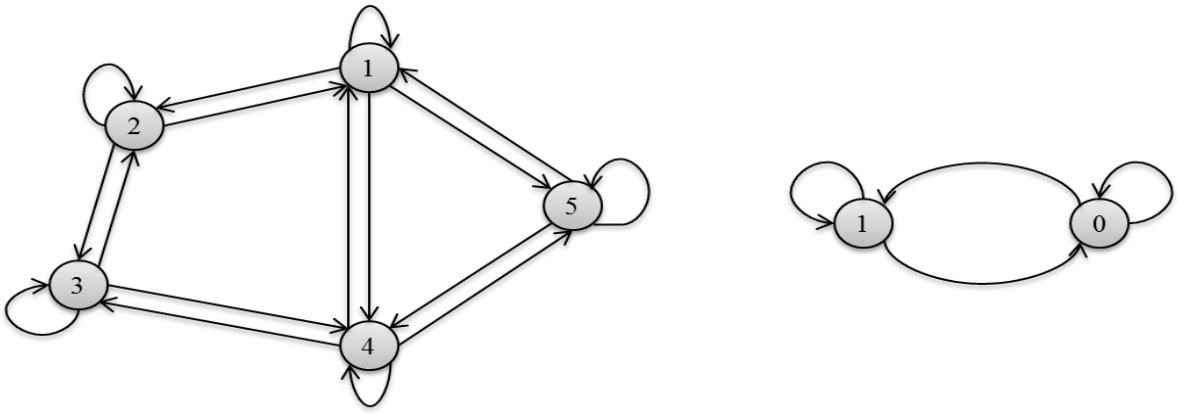

Figure 11: Examples for graphs of nodes and edges

Specifically, let us consider a graph of a finite vertex set V and a set of directed edges. Each vertex is associated with a positive mass of its own and each edge is associated with a positive weight. This can be denoted as a finite vertex set $\vartheta \epsilon V$ each with a positive mass $mass_{\vartheta}$. In addition to the vertices, this graph also comprises a set of positively weighted directed edges. For each of the orderly pair of distinct vertices ϑ_i and ϑ_j we associate a weight, i.e., $weight_{ij}$. Note that if $weight_{ij} = 0$ then a directed edge is not present while $weight_{ij} > 0$ shows that a weighted edge is present.

In this thesis I have considered a partitioning problem so as to classify the finite n vertices into m disjoint non-empty subsets so as to minimize a cost function, while possibly enforcing one or more constraints.

4.2 The Influence Model

The influence model is a network of interacting Markov Chains, with each chain being associated with a vertex site of the network and with the chains being allowed to differ from one site to another. The evolution of each chain is not just influenced by its own present status but also by the statuses of the chains at each neighboring sites on the network. All the nodes or

vertices, or the sites, are updated simultaneously at each time step. Every node is associated with a status. These statuses associated with each of the nodes form patterns as they evolve with time which eventually results in good partition(s). The influence model is claimed to have a rich mathematical structure and may provide a fruitful way of representing and studying a variety of dynamics [2].

The influence model is a network of n nodes or vertices or sites. Each of the sites is capable of taking a status $s_i[k]$ of the possible finite m number of possible statuses. The model is updated at each discrete time step k. We refer to the statuses of all sites at each time step.

Generally, each site i picks a site j as its determining site with a probability p_{ij}. The next status of the site i is determined with respect to the current status of the determining site j. The status of the site i is determined probabilistically, depending on the current status of the determining site.

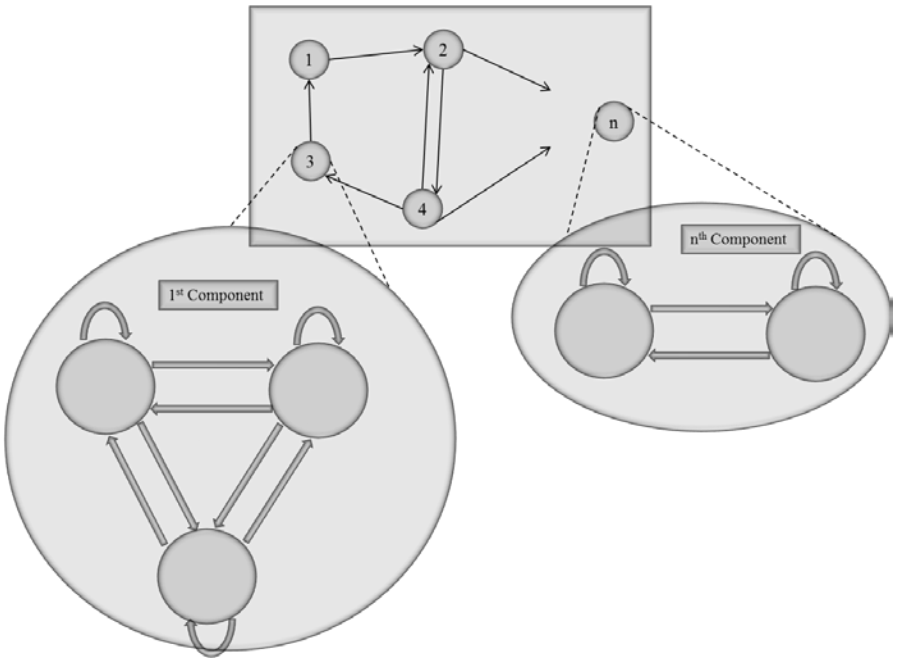

Figure 12: The influence model

4.2.1 Special Case of the Influence Model

I have used a special case of the influence model, called the copying influence model, developed by Y. Wan et al [1]. The model that I use evolves in discrete rather than continuous time, so the structure of the network is focused on determining whether all sites reach a consensus state. The notations used in this section are the same as those of the previous section.

Each site takes on the same number of *m* statuses. At each time step, each site merely copies the status of its determining site. As a recursive algorithm, at each time step, each site i picks a neighbor, j , or itself, with a probability p_{ij} and copies the current status of that neighbor or of itself. It is noted that the model requires interaction along with the immediate neighbors only and one can infer that the algorithm is decentralized in nature.

4.3 Algorithm Description

I have used the copying influence model as a tool to solve this partitioning problem. The copying influence model can be outlined as:

1) Mapping and formulation 2) Recursion 3) Termination

4.3.1 Mapping and Formulation

We map the graph to the copying influence model with m possible statuses to acquire m number of partitions at the end of the iterative algorithm. The mapping of the graph is such that large influences are associated with strong interconnections and weak influences with weak interconnections. However, asymmetric interconnection strengths are permitted. To map the graph to the copying influence model we need the copying probabilities. These indicate the branch weights between the connected sites as shown in the Figure 13.

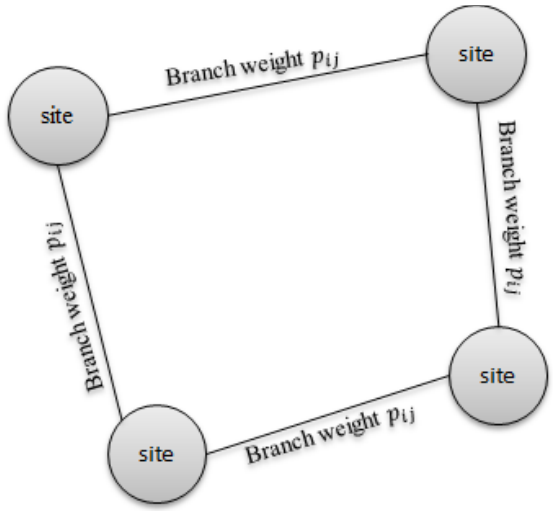

Figure 13: Distance between sites is then converted into branch weights

The branch weight p_{ij} can be calculated as

$$p_{ij} = \begin{cases} \dfrac{\Delta weight_{ji}}{mass_i}, & i \neq j \\ 1 - \Delta \sum_l \dfrac{weight_{li}}{mass_i}, & i = j \end{cases}$$

….. (8)

where Δ is chosen as: $\Delta \leq \frac{1}{\max\limits_i \sum_l \frac{weight_{li}}{mass_i}}$

….. (9)

This implies that large weights are associated with large influences and small weights are associated with small influences. In upcoming sections I discuss implementation on formulating and mapping the network practically.

4.3.2 Recursion

We choose the initial state for the copying influence model. The statuses of each site are updated iteratively, according to the developed copying influence model. Hence, a sequence of possible partitions of the graphs is generated. Since this is a distributed computation, the sites use information only from the immediate neighbors. I have found it convenient to consider the case of self-partitioning with reference nodes. Since the network has sites each with a status associated with them, I have implemented a slightly modified algorithm [1]. There is one reference node per status. Hence, for the problem of k-way partitioning algorithm, we fix k reference sites; in this case m reference sites, with distinctive or recognizable statuses from *0* to *m-1*. The reference nodes are assigned with distinctive statuses, while the rest of the sites are initialized with random statuses. The reference site always chooses itself as its determining site. A little modification to the influence model developed in the above equation (8) prompts the reference site to always choose itself as its determined site:

$$p_{ij} = \begin{cases} 0, & i \neq j, \\ 1, & i = j\,, \end{cases}$$

….. (10)

Since the partitioning is distributed in manner, we only require that the reference nodes know their own identities to implement this initialization. We then update the copying influence model. The state at each time step of the recursion offers a partition of the graph.

Note that:

1. The partition identified at each time step of the recursion automatically satisfies the set inclusion constraints for the *k*-way partitioning with reference nodes.

2. The recursion generates a random sequence of partitions and eventually finds an optimal solution with probability 1 under a broad assumption, after a certain while.

Moreover, the recursion is completely distributed. For simplicity, we assume that the agents have a common clock. Each agent in the network only needs to randomly select a neighbor at each time step, so the communication, at each time step, does not increase with the size of the network. The recursion generates a random sequence of partitions, until it finds an optimal solution with probability 1 which then concludes to m number of partitions.

Our next concern is to stop the algorithm. To have good stopping measures the algorithm should: (1) Select or identify low cost partitions quickly (2) Permit identification of the optimal solution with probability 1, under broad conditions.

As mentioned earlier, weakly connected sites do not influence each other and tend to maintain distinct statuses. The influence strengths are nothing but the edge weights and node masses. Thus a good partition can be considered in the model, where the strongly connected subsets have weak cuts between them. Thus, we can expect the copying influence model to find weak cuts between the stronger links and conclude an optimal partition. The following section discusses the stopping criterion used in the algorithm.

4.3.3 Stopping

The algorithm proposes to stop the updating when the minimum cost partition has not changed for a certain number of algorithm stages. The strategy is to progressively reduce the influence between sites with different statuses at each time step and after each update, such that there is a partition. Particularly, we gradually reduce the influence between the sites with distinct statuses after each update as we correspondingly increase the self-influence, simultaneously.

Mathematically, the time varying influence $p_{ij}[k]$ is modifies a follows:

– If $s_i[k] \neq s_j[k]$ and $p_{ij}[k] \geq \Delta$,

then $p_{ij}[k+1] = p_{ij}[k] - \delta \quad (i \neq j)$ and $p_{ii}[k+1] = p_{ij}[k] + \delta$;

….. (11)

– If $s_i[k] \neq s_j[k]$ and $p_{ij}[k] < \Delta$,

then $p_{ii}[k+1] = p_{ii}[k] + p_{ij}[k]$ and $p_{ij}[k+1] = 0 \ (i \neq j)$;

….. (12)

– If $s_i[k] = s_j[k]$,

then $p_{ij}[k+1]$ will remain the same.

….. (13)

The algorithm promises to find a good partition. This is because the weak edges in the initial graph tend to have different statuses at their ends in the influence model. Hence, these edges are removed by the algorithm. This strategy is referred as partitioning with adaptive stopping.

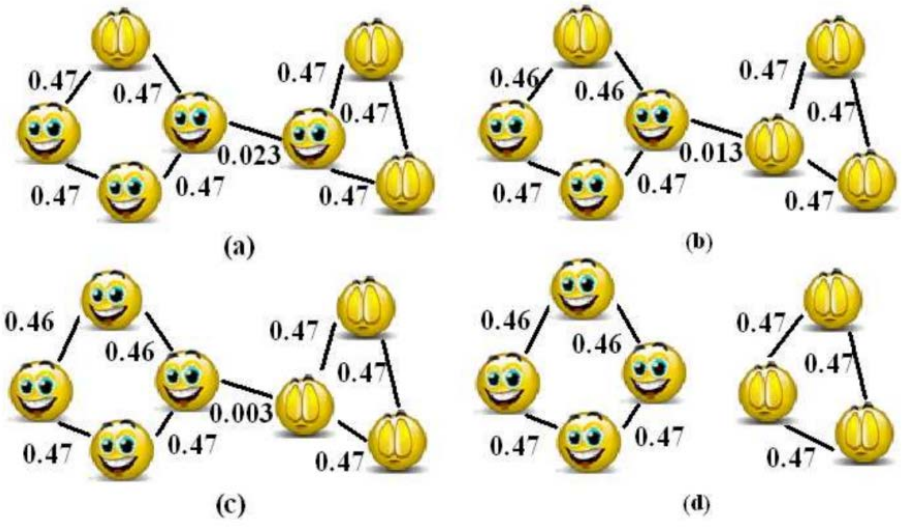

Figure 14: Illustration of the influence model, Source: [1] Permissions: [1]

4.4 Programming Platform

4.4.1 MATLAB® Implementation

To test the feasibility and the potential of the copying influence model, I present a MATLAB® testing environment. MATLAB® is an analytical tool; a numerical computing environment developed by MathWorks. MATLAB® is a commonly used program for computer modeling. It, normally, allows the implementation of algorithms, plotting functions of data, and matrix manipulations. Its code is relatively straightforward, so even though one may not use MATLAB®, and its pseudo-code flavor should be easy to translate into any other programming language.

In this testing environment I present results that prove the copying influence model finds a good partition, i.e., identifies an optimal solution with probability 1, given that the optimal solution satisfies certain broad connectivity conditions. Although the general case can find given number of k partitions for any finite n number of sites, it is necessary to note the value of δ. The δ will depend on the value of the branch weights, i.e., the p_{ij}.

The following series of figures show a partition for a five node network where $m = 2$, i.e., the nodes can be associated with only one of the two statuses 0 and 1. We recall that the influence model always has one reference node for each status. The results may vary for different values of δ.

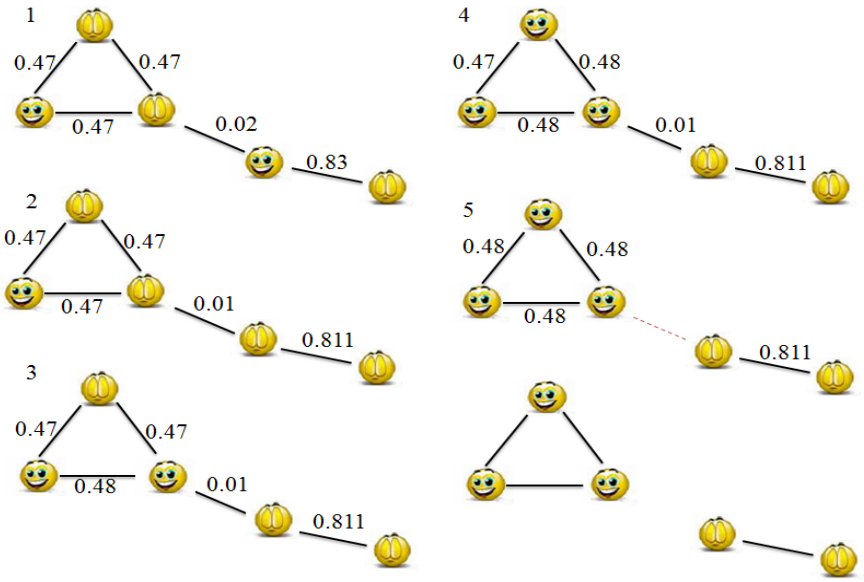

Figure 15: Copying influence model for five nodes with five numbers of iterations

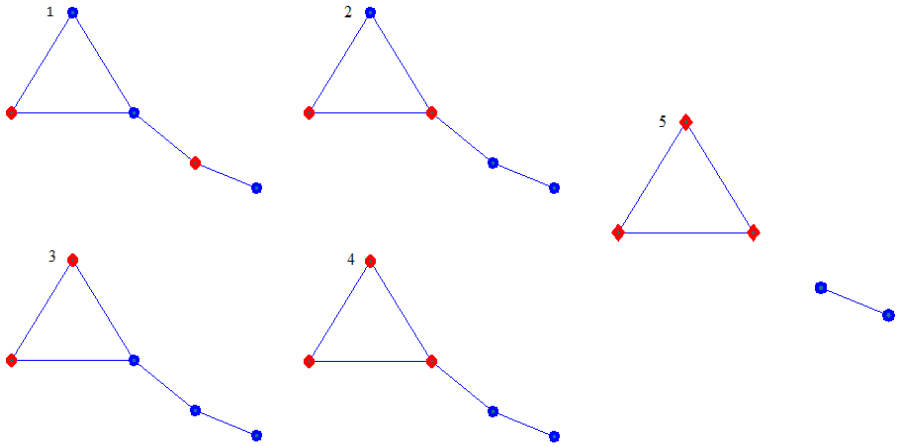

Figure 16: MATLAB® illustration for five nodes representing an instant of an example

Additionally, I have created a MATLAB® setting where one can input any number of nodes, their locations, and their initial statuses to analyze the copying influence model. The MATLAB® setting shows the framework with all the iterations carried out to conclude a partition. One can refer to [30] to watch the iterative framework of five nodes with two different statuses and gradually achieving a partition.

4.4.2 LabVIEW™ Implementation

LabVIEW™ was used as a tool to program the NXT devices as all the agents (Refer to section- 2.3). My system partitions the network formed after the agents announce discovery of the targets to their immediate neighbors and exchange the location information to form interconnect for the network. The flexible stochastic partitioning algorithm implementation is comprised of the following stages: mapping, initialization, partitioning, and stopping. The block diagram below suggests implementation on an agent level.

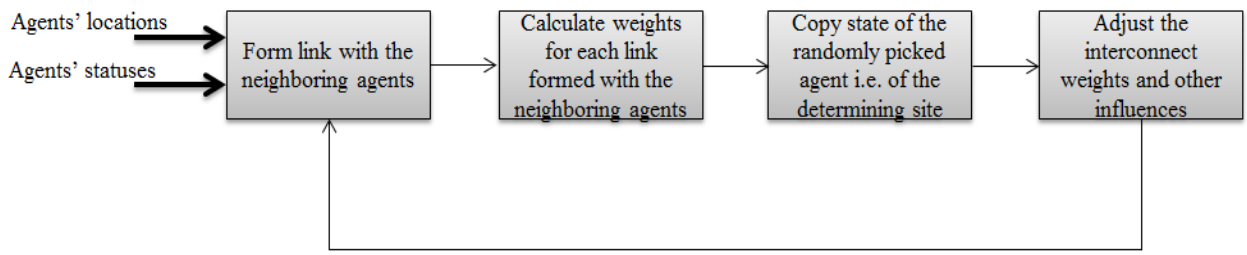

Figure 17: Distributed partitioning on agent level

Agents that discover target become reference agents (like the reference nodes). The rest of the agents, copying agents, are influenced by the reference agents. At the end of the partitioning algorithm, when there is a successful partition, a new planned trip is initiated with a final destination that matches to that of the target.

Let us take a look at the stages of the partitioning algorithm implemented in the LabVIEW™.

Mapping:

The formation of the network is explained in the previous chapters. I formed links between the agents based on the distances with the other agents, which helped to define an agent's immediate neighbors.

As a first step towards building the partitioning algorithm, we map a graph to the influence model such that the copying probabilities reflect the distances between the agents. A farther distance is unfit for a rendezvous, so the weights or the copying probabilities between the two agents (sites) have an inverse relationship with the corresponding distance.

$$distance\ between\ the\ agents \propto \frac{1}{p_{ij}} \qquad \ldots\ldots (14)$$

We refer to the Metropolis theorem [26] [27] to calculate the weights or the copying probabilities, given the distances between each node. This can be calculated as the following:

$$p_{ij} = \begin{cases} \frac{1}{(1 + max\{p_i, p_j\})}, & i \neq j; \\ 1 - \sum_{\vartheta \epsilon V} p_{i\vartheta}, & i = j; \\ 0 & otherwise, \end{cases} \qquad \ldots\ldots (15)$$

Recursion:

To develop an algorithm for a 2-way partitioning algorithm with reference nodes, we refer to two reference sites 0 and 1. We choose the initial status of the sites arbitrarily. Once the distances between the linked agents are calculated, we determine the copying probabilities. Since the probabilities are inversely proportional to the relative neighboring distances, for the neighboring agents that are at a farther distance, the copying probability between the agents is smaller. In this stage, i.e., the iteration stage, every agent updates its state by copying the status

of the neighbors, according to the copying probability of that neighbor. The agents begin the process with a low probability of keeping their status, and a subsequently high probability of being influenced by the neighboring nodes. This influence is dependent on the copying probabilities. Nodes with better connectedness have a better chance of spreading their statuses.

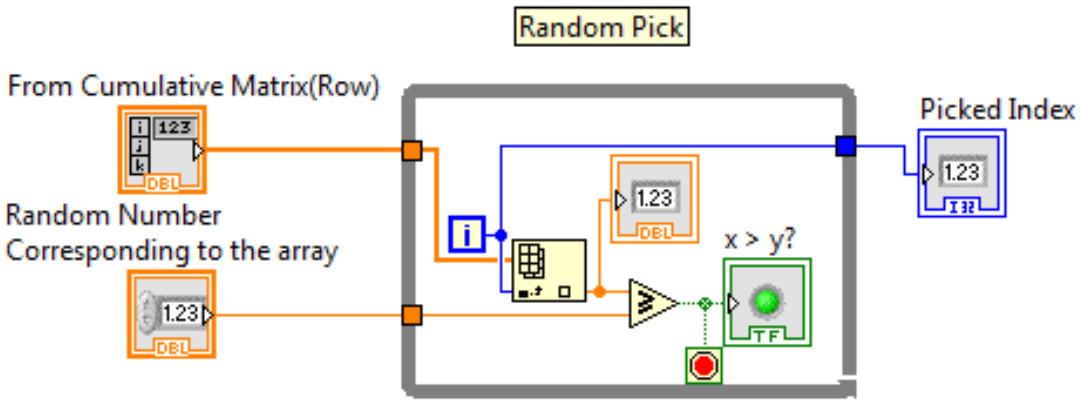

Figure 18: A snapshot of the algorithm for randomly picking the neighbor based on the influence of the copying probability

Stopping:

Initially when the algorithm starts, there may or may not be weak copying probabilities (the agents at a relatively farther distance). Nevertheless, the goal is to produce these poor connections so that the poorest connection ultimately decays below the threshold δ, thus partitioning the network. One needs to be careful while choosing δ such that it can be less than all of the possible connected copying probabilities. It is proved in [1] that the smaller the δ the better is the partition with probability 1.

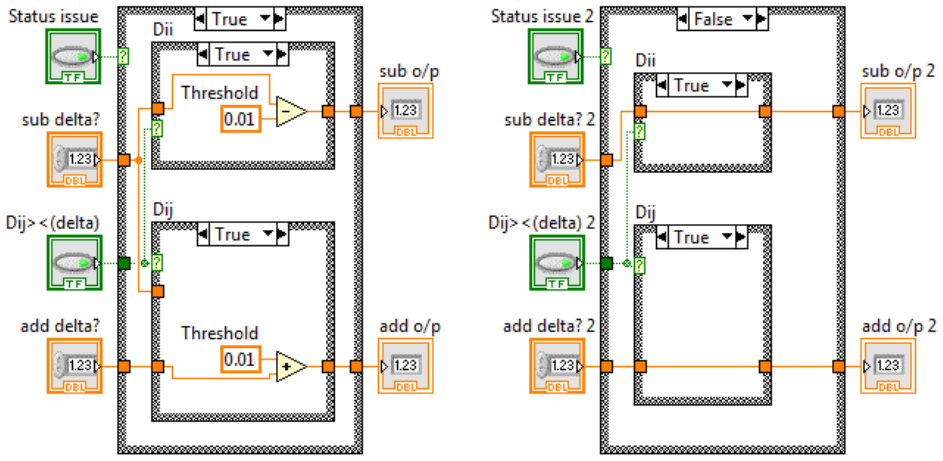

Figure 19: LabVIEW™-snapshot showing the stopping criteria for the distributed partitioning

4.5 Test Bench Set up

The same arrangement of the agents with a network formed (as discussed in the previous chapters) is continued in the distributed partitioning. Once all the targets were discovered, partitioning began. Agents that found the targets became the reference nodes and always chose themselves as their determining site (the reference site always chooses itself). Moreover, the copying agents (rest of the agents in the network) are then influenced by these nodes to encourage an optimal partition. Because all the reference agents had distinct statuses, which were nothing but at the target location, the sub-sequential planned trip for the copying agents was

towards the respective reference agents. Note that all of the agents are more or less identical.

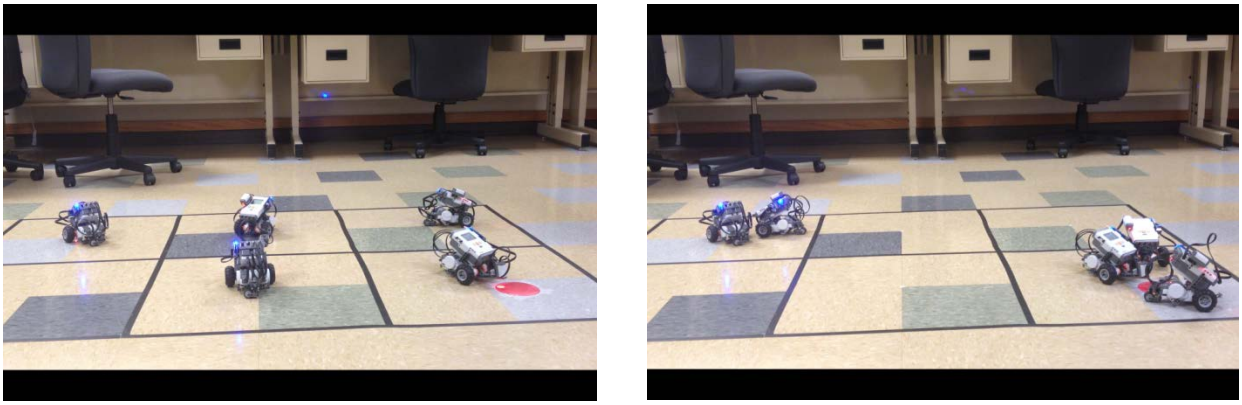

Figure 20: A snapshot of the algorithm, showing the formed network and the final trip of the copying agents to the reference agents.

4.6 Communication

I have used the NXTBee radio module for exchanging information about the statuses. The statuses of the copying agents are updated iteratively. For reiteration, the agents require the updated statuses of their immediate neighbors. Since all the agents are independent, each is capable of extracting status information from its neighbors.

I developed an environment where each agent sends an acknowledgement after the reception of the information of the statuses. Importantly, the stopping of the algorithm is essentially decentralized in nature. When the weakest of the interconnect decays below the threshold value, an agent broadcasts a signal and the recursion ends. Just as the recursion ends, the copying agents plan a trip to their final destinations which are merely locations of the reference nodes.

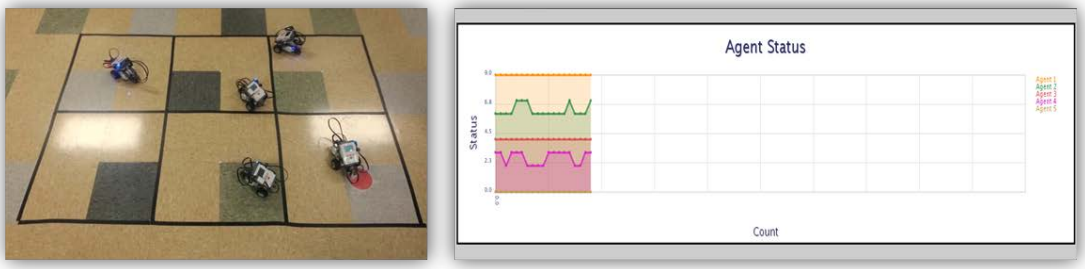

Figure 21: Example of a network carrying out iterations for distributed partitioning and its real time mapping on the computer via Xbee module

To verify the real time mapping of the copying influence model an XBee model is connected to the computer and the influence iterations are mapped w.r.t. to the statuses. The figure 21 shows the snapshot of real time mapping corresponding to the formed network.

Figure 22: Series of snapshots showing the partition and the trip to the final destination expressing partition

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Introduction

This chapter displays graphical and pictorial results of the random exploration and distributed partitioning. I prove that with the use of low cost NXT devices and a simple but reliable communication mode it is possible to have a practical environment of a mobile automaton which is self-tracking and self-organizing in-order to stimulate a steady state. I also discuss the futuristic potentiality of the automaton and its possible applications.

5.2 Results and Analysis

The results mainly depend on two prospects: 1) the accuracy with the heading and forward movement of the agents, i.e., the NXT devices and 2) the test-bed's ability to find an optimal partition with probability 1 with the formed network after the target acquisition. In this section I analyze the implementation results of the test bench with NXT devices that showcase target acquisition through random exploration and distributed partitioning algorithm.

Physically, the most common challenges during the implementation were 1) the accuracy of the motors corresponding to the heading changes, 2) the accuracy of the sensors to sense the targets and the 3) synchronization of the communication. In random exploration, when tracking became prolonged the use of the proportional controller for the heading and forward movement accumulated an error. The rotational heading adjustment Sub-VI results in a heading accuracy within $\pm 5°$. Moreover, unlike in most of the NXT movements performed till now, I have saved the time required for the rotation (by rotating both the wheels in opposite direction to each other). I have run, in parallel, the tracking of the targets (with a light sensor), with a

multithreading capability of the LabVIEW™ used. The figure below shows the real time movement of an agent. Note that the tracking is more or less uniformly distributed.

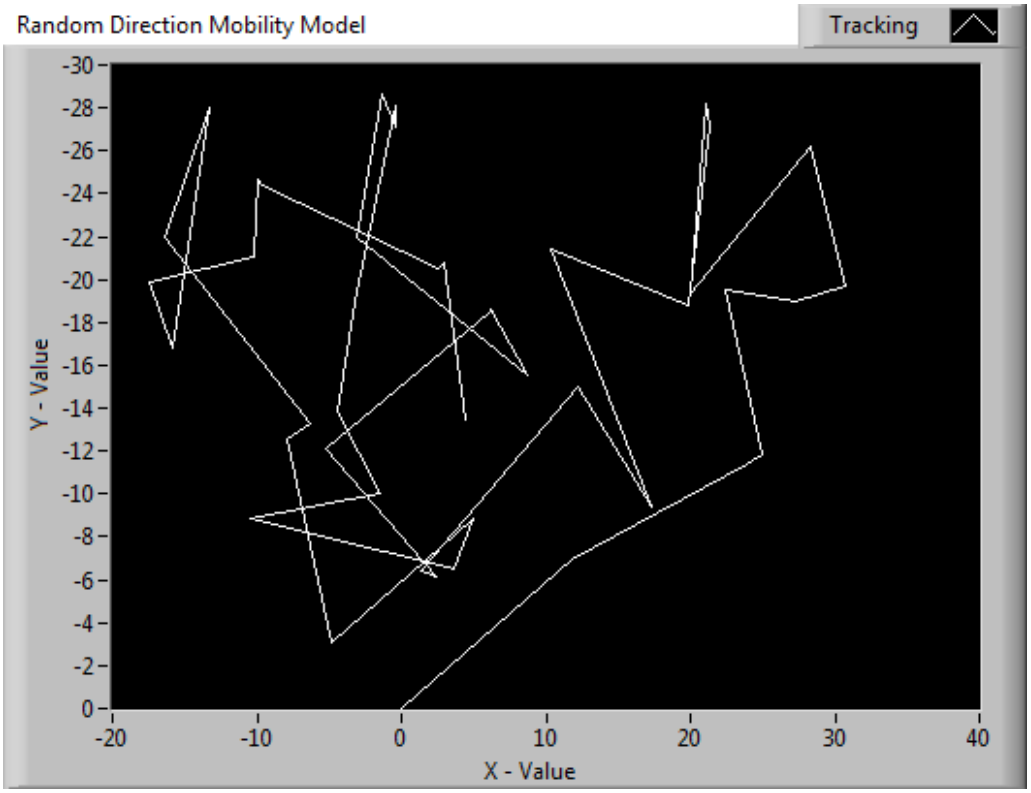

Figure 23: Real time random movement of an agent for random exploration

The partitioning algorithm did not encounter a lot of physical challenges, although a good partition was entirely dependent on the network links that were formed. Nevertheless, a partition would be accomplished with the gradual weakening of the least probable link. The agents were required to converge to a reference point after the partition. It was then noticed that the more the time taken for random movement of the agents to track a target, the more the agents deviated to reach to their final reference destination. But this deviation entirely depended on the time taken for the random mobility, which was, in turn, because of the accumulation of the heading error.

Another criteria for the fast convergence of the agents to have a steady state depended on the δ. We choose δ such that we find the optimal partition with probability 1 and also want to have a steady state as quickly as possible. The lower the δ, the more the number of iterations but the better is the performance. Shown below is an example of a five-node network to perform the distributed partitioning. In this example the partition takes place when the weakest link goes below the δ.

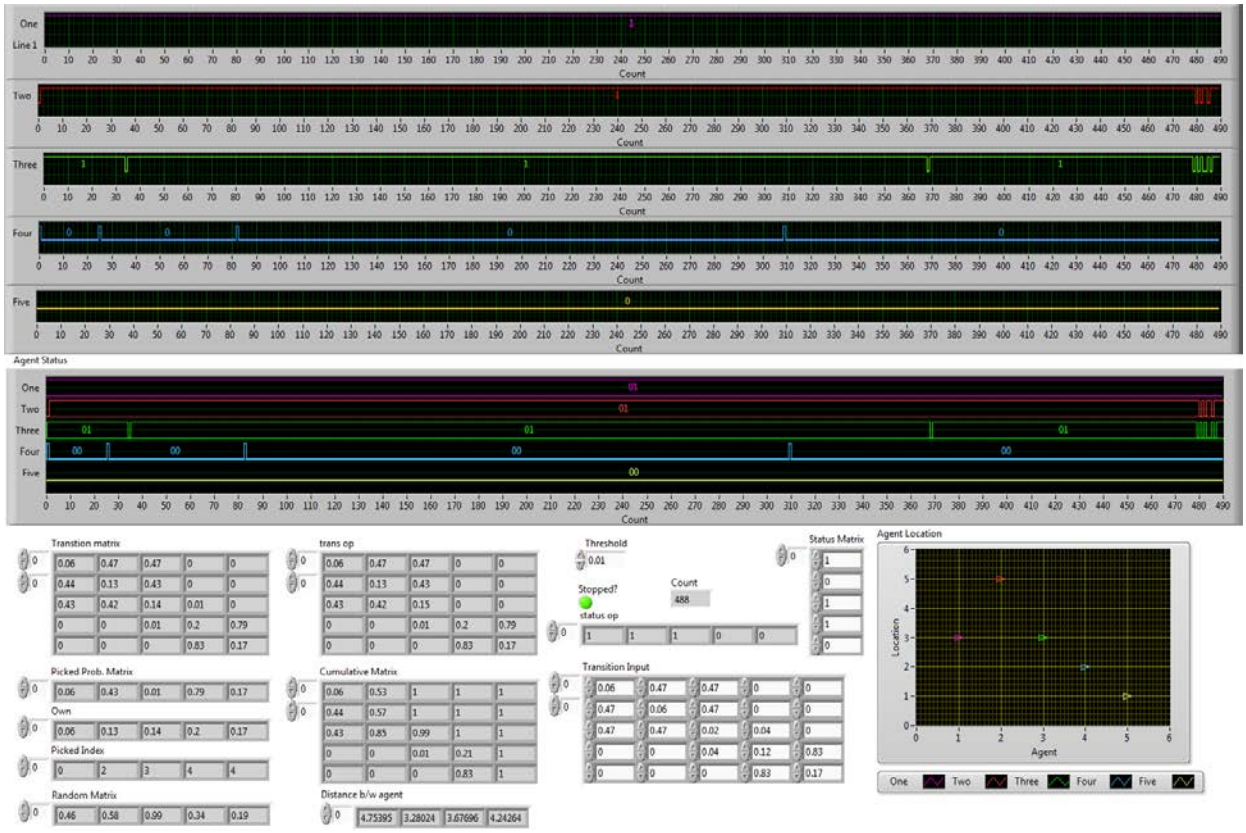

Figure 24: Results of the distributed partitioning algorithm with a 5 agent network

I demonstrate a decentralized robot swarm for randomized exploration and distributed partitioning based on a local data and realistically constrained network architecture. The communication with the help of the NXTBee radio module is slow but accurate. I have compromised with the operation speed in order to provide considerable accuracy.

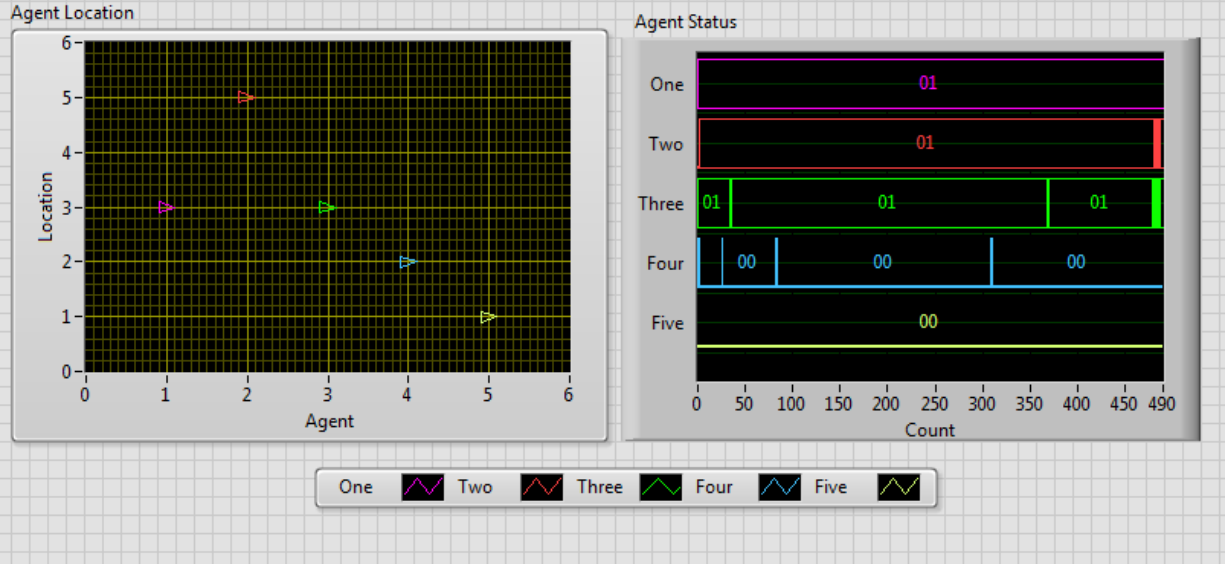

Figure 25: A magnified view of the convergence of the five agents in a stable state

5.3 Conclusions and Discussions

This thesis has described what I have termed as the ‘A LEGO® MINDSTORMS® NXT based test-bench for Multi-Agent Exploratory Systems and Distributed Network Partitioning’. The LEGO® MINDSTORMS® NXT kits prove to be a prototypical platform to present a stand-alone unit capable of tracking and organizing itself to come to a steady state. My model is generalized and relates to other models of the networked stochastic automatons. The distributed nature of the algorithm is my model’s important feature. The generalization demonstrated could prove to have important degrees of freedom in particular applications.

5.4 Future Possibilities

My test-automaton, on a generalized basis, performs a rendezvous task and can be applied to execute decentralized formation maneuvers. The test bed presents a practical prototype that can be implemented in several applications with the UAVs, drones, mobile ad hoc networks (MANET), and in military and civilian applications, and so on.

As a part of further modifications, one can use a better controller, like a proportional-integral or proportional-integral-differential controller for better accuracy and smoother mobility. Additionally, advanced mobility patterns can be implemented that mimic the practical behavior of ground or aerial vehicles.

I have used the LEGO® MINDSTORMS® NXT for the multi-agent exploration. The NXT Light Sensor I have used is quite sensitive and precise. It reads light intensity from the surrounding environment, as well as the reflection from the infrared transmitter. The LED light source can be turned off for an accurate ambient light reading. Also, the motors used have an encoder count of $360°/\text{revolution}$. One can shift to motors with better encoder resolution, yielding sharper and smoother mobility. For tracking, one could use more advanced odometry methods and sensors with better sensitivity and precision.

In the case of distributed partitioning, the selection of reference vertex sets required in my test bed is objectionable when a large number of partitions are needed. My further amendment involves in reference free partitioning methods. The time required for the partitioning depends on the formation of the network and δ chosen to partition with respect to the weakest influence. The time required to find an optimal partition increases with the reduction in δ. I am exploring strategies to initialize the agents with different states and to keep updating till the desired number of partitions is identified. There can be further development with considering unexpected events while navigating a given field.

My test bench has a rich mathematical structure. It can represent and study a variety of issues in network dynamics. My test-bench, thus serves as a basis for developing more complicated models better able to characterize the essentials in applications of interest, along with several other dimensions to be explored.

BIBLIOGRAPHY

[1] Y. Wan, S. Roy, A. Saberi, B. Lesieutre, "A stochastic automaton-based algorithm for flexible and distributed network partitioning," *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE* , pp. 273-280, June 2005.

[2] C. Asavathiratham, S. Roy, B. Lesieutre, G. Verghese, "The influence model," *Control Systems, IEEE* , vol. 21, no. 6, pp. 52-64, Dec 2001.

[3]V. Sheth, M. Bokshi, Y. Wan, "A test Bed for Multi-Domain Communication Networks using LEGO Mindstorms," i*n proceedings of AIAA Infotech@aerospace Conference,* August 2013.

[4] Y. Kim, "Control Systems Lab Using a LEGO Mindstorms NXT Motor System," *Education, IEEE Transactions*, vol. 54, no.3, pp. 452-461, August 2011.

[5] D. Miller, "Multiple behavior-controlled micro-robots for planetary surface missions," *Systems, Man and Cybernetics, 1990. Conference Proceedings, IEEE International Conference*, pp. 289-292, November 1990.

[6] S. Sail, S. Yang, "On the Applicability of Random Mobility Models for Swarm Robot Movements".

[7] N. Maze, Y. Wan, K. Namuduri, M. Varanasi , "A Lego Mindstorms NXT-Based test bench for Cohesive Distribute Multi-agent Exploratory Systems: Mobility and Coordination," *in proceedings of AIAA Infotech@aerospace Conference*, 2012.

[8] S. Brigandi, J. Field, Y. Wang, "A LEGO Mindstorms NXT based multirobot system," *Advanced Intelligent Mechatronics (AIM), IEEE/ASME International Conference*, pp. 135-139, July 2010.

[9] M. Habib, H. Asama, Y. Lshida, A. Matsumoto, I. Endo, "Simulation Environment For An Autonomous And Decentralized Multi-agent Robotic System," *Intelligent Robots and Systems, in proceedings of the IEEE/RSJ International Conference,* vol. 3, pp.1550-1557, July 1992.

[10] W. Ren, R. Beard, E. Atkins, "Information consensus in multivehicle cooperative control," *Control Systems, IEEE Transactions*, vol. 27, no. 2, pp. 71-82, April 2007.

[11] W. Ren, R. Beard, T. McLain, "Coordination Variables and Consensus Building in Multiple Vehicle Systems," *in proceedings of the Block Island Workshop on Cooperative Control, Springer-Verlag Series: Lecture Notes in Control and Information Sciences*, vol. 309, p. 171-188.

[12] V. Genovese, P. Dario, R. Magni, L. Odetti, "Self Organizing Behavior And Swarm Intelligence In A Pack Of Mobile Miniature Robots In Search Of Pollutants," *Intelligent Robots and Systems, lEEE/RSJ International Conference*, vol.3, pp. 1575-1582, July 1992.

[13] J. Cheng, W. Cheng, R. Nagpal, "Robust and self-repairing formation control for swarms of mobile agents," *AAAI*, vol. 1, pp. 59-64, 2005.

[14] J. Lin, A. Morse, B. Anderson, "The multi-agent rendezvous problem," *Decision and Control, 2003. Proceedings. 42nd IEEE Conference,* vol. 2, pp. 1508-1513, December 2003.

[15] G. Kantor, S. Singh, R. Peterson, D. Rus, A. Das, V. Kumar, G. Pereira, J. Spletzer, "Distributed Search and Rescue with Robot and Sensor Teams," *The 4th International Conference on Field and Service Robotics*, July, 2003.

[16] Y. Meng, J. Gan, "A distributed swarm intelligence based algorithm for a cooperative multi-robot construction task," *Swarm Intelligence Symposium, IEEE Transactions*, pp. 1-6, September 2008.

[17] J. Fax, R. Murray, "Information flow and cooperative control of vehicle formations," *Automatic Control, IEEE Transactions*, vol. 49, no. 9, pp. 1465-1476, September 2004.

[18] S. Roy, A. Saberi, K. Herlugson, "A control-theoretic perspective on the design of distributed agreement protocols," *American Control Conference*, vol. 3, pp. 1672-1679 vol. 3, June 2005.

[19] A. Purnamadjaja, R. Russell, "Pheromone communication: implementation of necrophoric bee behaviour in a robot swarm," *Robotics, Automation and Mechatronics, IEEE Conference*, vol.2, pp. 638-643, December 2004.

[20] J. Lin, A. Morse, B. Anderson, "The multi-agent rendezvous problem," *Decision and Control, in proceedings of 42nd IEEE Conference*, vol. 2, pp. 1508-1513, December 2003.

[21] J. Cortes, S. Martinez, F. Bullo, "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions," *Automatic Control, IEEE Transactions*, vol.51, no.8, pp. 1289-1298, August 2006.

[22] J. Lawton, R. Beard, B. Young, "A decentralized approach to formation maneuvers," *Robotics and Automation, IEEE Transactions*, vol. 19, no. 6, pp. 933-941, December 2003.

[23] LabVIEW™ Toolkit for LEGO® MINDSTORMS® NXT Programming Guide, Getting Started with LabVIEW™, http://www.ni.com/manuals/

[24] P. Nain, D. Towsley, B. Liu, Z. Liu, "Properties of random direction models," *INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, pp. 1897-1907, March 2005.

[25] G. Ravikiran, S. Singh, "Influence of mobility models on the performance of routing protocols in ad-hoc wireless networks," *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, vol.4, pp. 2185-2189, May 2004

[26] Y. Wan, K. Namuduri, Y. Zhou, S. Fu, "A Smooth-Turn Mobility Model for Airborne Networks," *Vehicular Technology, IEEE Transactions*, vol. 62, no.7, pp. 3359-3370, September 2013.

[27] B. Cheng, R. Charland, P. Christensen, L. Veytser, J. Wheeler, "Evaluation of a Multi-hop Airborne IP Backbone with Heterogeneous Radio Technologies," *Mobile Computing, IEEE Transactions*, vol. 13, no. 2, pp. 299-310, February 2014.
doi: 10.1109/TMC.2012.250

[28] R. Olfati-Saber, J. Fax, R. Murray, "Consensus and Cooperation in Networked Multi-Agent Systems," *in proceedings of the IEEE* , vol. 95, no.1, pp. 215-233, January 2007.

[29] K. Shoemake, “Animating Rotation with Quaternion Curves,” SIGGRAPH 85, *Proc. Computer Graphics*, vol. 19, no. 3, pp. 245-254, July 1985.

[30] Matlab illustration link: https://www.dropbox.com/s/miassohud13sux1/MatlabMovie.wmv

[31] Real-time illustration:
https://www.dropbox.com/s/y504viwizzy3sms/Video%20Illustration.MOV
https://www.dropbox.com/s/mhpxhnd1kehh9m0/Mapping%20with%20Illustration.wmv
(To play the videos together)