A Behaviour-Based Hierarchical Communication Strategy for Multi-Robot Systems

By

Zhengzheng Wan

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

December 18, 2007

Your file Votre référence
ISBN: 978-0-494-36842-8
Our file Notre référence
ISBN: 978-0-494-36842-8

Abstract

Recent work in the design and implementation of multi-robot systems has focused on hierarchical arrangements and communication strategies for such systems. We present here a new hierarchical communication strategy for multi-robot systems that uses a bottom-up approach. The system uses neuron networks to maintain robot communication, to assign to robots implicit work zones, to navigate towards various static/dynamic goal locations and to perform any necessary task-related behaviors. This approach results in a faster and more reactive system than more traditional approaches and also reduces the computational power requirements. Our strategy ensures that each robot remains within communication range of its group leader at all times and can also be generalized to multi-level hierarchies. We simulated the system and then performed experiments that show how a single-level hierarchical system outperforms its non-communication counterpart in terms of concentrated area coverage when performing path-directed tasks and when focusing on centralized search patterns.

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. Firstly, I deeply thank my supervisor Dr. Mark Lanthier for his continuous, patient, various advice and help. I am happy to work with him on interesting projects. Being a smart and rich-experienced person, Mark always pointed out good ideas. Although English is not my first language, Mark always kept the high passion to help me to improve my writing skill.

I also thank Dr. Doron Nussbaum for his very meaningful comments and suggestions on my thesis.

Tremendous thanks must be given to my parents, Jiacong Li and Yongzheng Wan, who I haven't seen for many years. My parents are the most respectable and special persons for me in the world, They always stand by my side to encourage me. I thank them for their both moral and financial support. I love them forever!

I also want to thank Yanxuan Zhang for her good friendship. Yanxuan and me are both working under the supervision of Dr. Mark Lanthier for three years, we always encourage and support each other to overcome the hardest time. The time we spent together at Carleton is wonderful!

Contents

List of Tables

List of Figures

Contents

Chapter 1

Introduction

1.1 Motivation

As shown by the increasing number of publications, the field of Multi-Robot Systems (MRS) has grown dramatically in recent years. Perhaps this is largely due to the fact that researchers have noticed that multi-robot systems provide advantages over single-robot systems. One main advantage is that, they can accomplish complex tasks more efficiently than a single robot. For example, [11] has demonstrated that multiple robots can localize themselves faster and more accurately by exchanging information about their positions whenever they can sense each other. Also a multi-robot system composed of low-cost robots is more fault-tolerant than a single robot system, making the overall system more robust. Since robots in MRS can execute tasks cooperatively, much less computational burden is given to each robot system in MRS than that given to a single robot system, because the decision making for each of those robots can be distributed. By exchanging, broadcasting and integrating information among robots, a MRS can handle more complex problems within a much

larger domain.

Through collective behaviors and cooperation, robots within a MRS can more efficiently perform difficult tasks than a single robot system. The applications of multi-robot systems are numerous. The most popular applications can be classified into five areas, they are *Object manipulation*, *Foraging*, *Exploration* and *Pattern formation*, and *Team competitions*.

- **Object manipulation**: Object manipulation is the general category of the problems in which multiple robots cooperatively manipulate large objects. One of the most well-known applications related to object manipulation is box-pushing.

- **Foraging**: Foraging problems require a group of robots to cooperatively find and possibly pick up objects scattered in the environment. It encapsulates a large class of practical problems such as cleaning, harvesting, searching, gathering and other coverage-based problems.

- **Exploration**: Exploration is the problem in which multiple robots move around in an environment, exploring unknown areas and collecting data. Through exploration, some particular tasks such as map construction and searching can be accomplished.

- **Pattern formation**: Pattern formation represents a class of problems in which robots in a MRS , move separately, but through collective behaviors, form one or more groups, where each group keeps a certain shape. Swarm-robotics, ant-like robots are the most representative works arising from pattern formation problems.

- **Team competitions**: Team competition is a class of applications in which a number of teams composed of completely autonomous robots, through cooperation among each team member, compete with one other. Team competition problems is mainly represented by robot soccer.

The level of cooperation within a MRS is usually dependent on the communication protocol between the robots. As the tasks become more complicated or the environment becomes larger and more dynamic, the inter-robot communication strategy becomes increasingly critical in order to maintain the efficient cooperation among the robots. There are three main types of communication architectures used in multi-robot systems [4].

- **Centralized**

  The centralized communication architecture is a traditional communication architecture which requires a central base station or a centralized control system to control and organize other robots. The central control system/robot is involved in the decision making for the whole MRS, while other robots act according to the directions of the "leader" [12]. In a centralized architecture based MRS, each robot communicates with the central base station by returning to the base station to exchange information.

- **Decentralized**

  In a decentralized communication architecture, the cooperating robots are responsible for making their own decisions. Instead of a centralized control system, each robot acts independently. Each robot in this architecture communicates with its neighboring robots through messages.

- **Hierarchical**

  A MRS with a hierarchical communication architecture allows one or more

robots to operate under the control of a "local leader", and also be the "local leaders" of other sub level robots. In such a communication structure, each robot must maintain communication with its "local leader" at all times. Hence, there is a local centralized communication group composed of a local leader and a number of its "worker" robots. The arrangement of local leaders can be done recursively, creating multiple levels of communication groups.

The communication among different groups is accomplished via their shared higher level local leader. Lowest level robots act as workers, while the top level robot acts as a super leader who is in charge of making decisions for the entire MRS. The use of the system may ultimately take on the role of super leader so as to allow human monitoring and/or intervention to ensure proper task completion.

This thesis presents a novel approach towards coordinating communication among robots in a multi-robot system which uses the Hierarchical Communication Strategy for Multi-Robot Systems (*HCSMRS*).

Figure 1.1 contrasts centralized and hierarchical communication architectures for a MRS. Figure 1.1A shows a single level hierarchical communication architecture for a multi-robot system where the robots are scattered in a environment. Each robot can only communicate with nearby robots or with the base station. Figure 1.1B shows a two-layered HCSMRS consisting of the same robots shown in Figure 1.1A as well as additional layer of local leaders. The ring around each local leader represents the safe communication range for the worker robots in that leader's group.

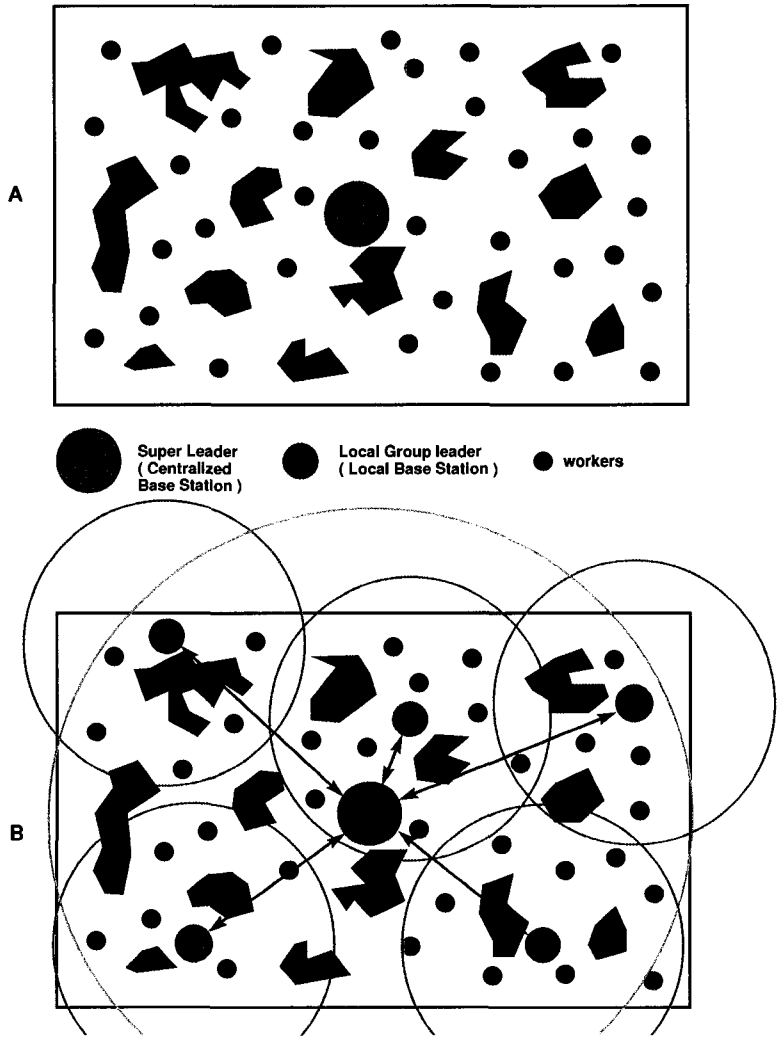


Figure 1.1: A: The single level hierarchical communication architecture for a MRS B: A two layer HCBMRS

Since a hierarchical communication architecture is composed of several local communication groups, each group of robots can move separately towards designated goals or areas through the motion of the group's leader. This makes it easier for the HCSMRS to distribute tasks to various groups as well as to distribute groups to areas that need attention in order to accomplish the overall tasks more efficiently.

Consider, for example, a situation in which a number of a robots in a MRS are required to concentrate their efforts (e.g. search, cleaning, map, etc,.) along a given path. Figure 1.2A shows how a group of robots in a HCSMRS can be led by their leader to explore along a given path $P_1 \rightarrow P_2 \rightarrow \cdots \rightarrow P_{12}$ in an obstacle-dense environment. Since the worker robots maintain contact with their leader at all times, they are implicitly directed along the same path $P_1 \rightarrow P_2 \rightarrow \cdots \rightarrow P_{12}$ while still having the flexibility to travel freely within their leader's vicinity. In order for the workers to still carry out their task assignments, it is necessary for the leader to move at a much slower pace than its group members.

The advantage that HCSMRS has over a distributed communication architecture becomes evident in such a path-directed task assignment. Lacking the ability to be directed globally, worker robots in a distributed architecture would cover the entire environment with equal attention as opposed to concentrating efforts along the important path $P_1 \rightarrow P_2 \rightarrow \cdots \rightarrow P_{12}$. It is true that, each robot in a distributed system can be given a similar fixed path to follow. However, in this case, each worker robot need to constantly monitoring their location, requiring them to have more sophisticated odometry systems. With the HCSMRS, the odometry mechanisms need not be as complicated because they can update their locations based on the leaders. In addition, with the HCSMRS, the robot remain in communication contact at all times, allowing the entire group of worker robots to be redirected to a new work site

at any time.

Another scenario that lends itself well to the HCSMRS architecture is in regards to searching for targets that are expected to be found at some central area in the environment. Robots in a MRS are required to find these targets in a limited amount of time. Figure 1.2B shows the situation in which a group of robots in a HCSMRS are executing a searching pattern. Robots begin by searching at some central location. Here, the leader may circle around the area while the workers perform the search. After some time, the leader may decide to expand the searching area by expending its own circling radius. As a result, the workers search a larger area. This can continue as long as necessary. The leader may itself move randomly with its own circle (Figure 1.3), or travel along the circle's circumference (Figure 1.5), or even spiral slowly outwards (Figure 1.7). The effect is either overlapping disk (Figure 1.4) or band-like (Figure 1.6 and Figure 1.8) type of search pattern by the workers.

The benefit of the HCSMRS under such a scenario is again evident when compared to the distributed or centralized architecture. The centralized architecture could easily accomplish an "overlapping disk" type of search pattern by simply informing all robots to search within a specific radius or along a specified circle, but again, the randomness would be lost. Also the HCSMRS allows various groups to simultaneously perform different search strategies with ease.

1.2 Behaviors and Mechanisms

In order to verify the feasibility of our HCSMRS system, we implemented it using a behavior-based framework. Behaviors are modules that cause the robot to perform

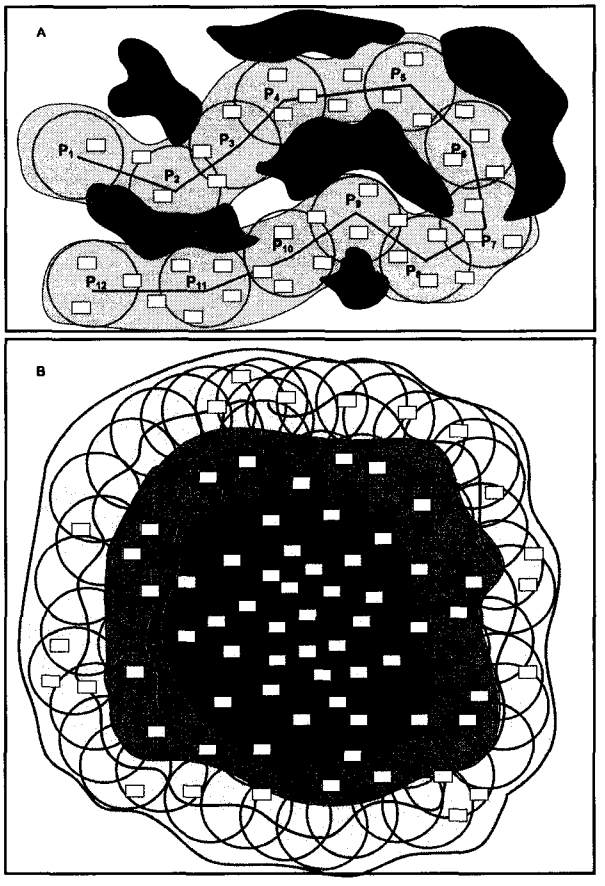


Figure 1.2: A: A group of HCSMRS robots are leaded to search along a given path. B: Band-like searching pattern.

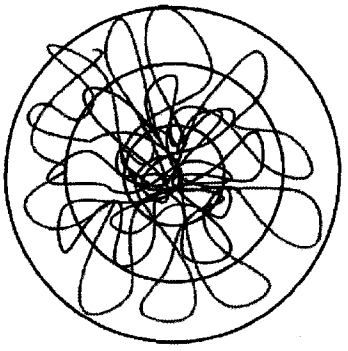


Figure 1.3: Outward overlapping disk searching pattern of a group leader.

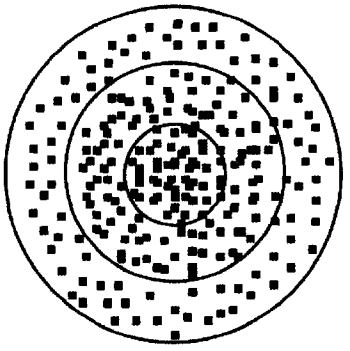


Figure 1.4: Effect of overlapping disk searching pattern generated by worker robots

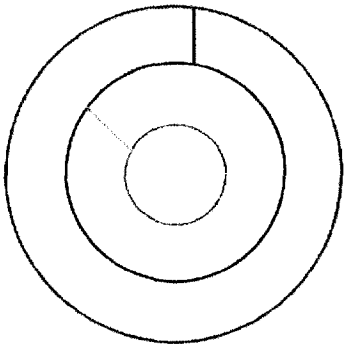


Figure 1.5: Outwards band-like searching pattern of a group leader

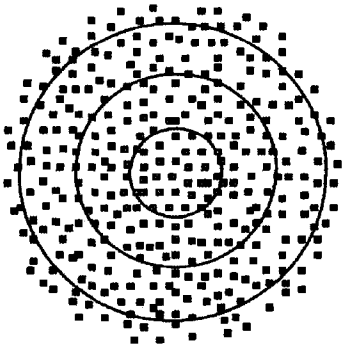


Figure 1.6: Effect of band-like pattern generated by worker robots

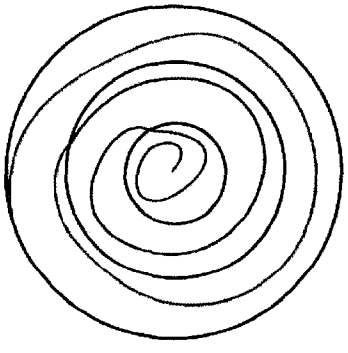


Figure 1.7: Outward spiral searching pattern of a group leader

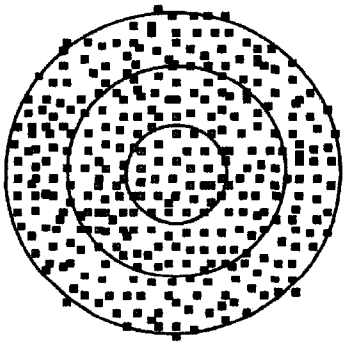


Figure 1.8: Effect of spiraling searching pattern generated by worker robots

simple pre-defined actions based on particular sensor input. Each of these behaviors connect together to produce an overall framework to control the robot's overall actions. We chose a behavior-based approach since it is a well accepted standard for robot control software and has proven to be quite robust when compared to traditional top-down approaches, which typically require the environment to be modeled beforehand as well as requiring very complex task plans [3]. In addition, the bottom-up nature of the behavior-based approach results in a more reactive system with quicker action, perhaps turning back on its path, to ensure that communication with the leader is not lost.

The framework concentrates on the basic movement of the robots, while ensuring that the communication hierarchy remains intact. We call it a framework because it allows future behaviors and mechanisms to be incorporated with ease, due to the plugable (i.e., modular) nature of the system. Figure 1.9 depicts the behavior-based system framework, showing the behaviors and control mechanisms that implement the HCSMRS and how they work together to achieve navigation and communication behaviors. Each behavior represents an independent process and all behaviors implemented compete for overall control of the robot's motion (i.e., they compete to steer the robot). The behaviors compete using a standard prioritized arbitration strategy. Higher priority given to behaviors that demand an immediate response (e.g., OutOfRange and Obstacle Avoidance) and lower priority is given to those with less demand on reaction time (e.g., Wandering and DirectToGoal Navigation). In addition to these behaviors, various control mechanisms have been implemented. These mechanisms are responsible for selecting behaviors as well as monitoring the robot's position.

The behaviors and mechanisms that control the robot's movements can be grouped into three main types:

1. **Low-level Movement**

   General movement behaviors that are common to general robot motion such as *Obstacle Avoidance* and *Wander*. Obstacle avoidance should be given high priority to prevent damage. Wandering is necessary to incorporate to a degree of randomness in the robot's movements and is often used to get the robot "unstuck".

2. **Communication-Related**

   Communication related behaviors that are responsible for keeping the robot within proper communication range with their leader. This includes the *Out-Of-Range Avoidance* behavior that turns the robot around when it is going too far out of the communication range, as well as the *Out-Of-Zone Avoidance* behavior that ensures that robots do not overlap one another's area while remaining within the communication range circle.

3. **Navigation-Related**

   Navigation related behaviors that direct a leader robot (and hence also its workers) either along a fixed trajectory or towards/away from a specific location (i.e., *Average Positioning Navigation*), *4-Vector Navigation* and *Combined Vector Navigation*). Their purpose is to steer groups of robots according to some desired strategy which is dependant on the overall task at hand.

The *Steering Mechanism* in Figure 1.9 is the arbitrator into which all behaviors are plugged. It decides, based on some fixed priority (which can be dynamically modified, but currently hard-coded in our implementation) which behavior will have

control of the robot at any specific time. Basically, each behavior requests to either steer the robot forward, left, right or not at all. The arbitrator simply accommodates the request of the behavior with the highest priority.

The *Odometry Mechanism* is used to measure and maintain the robot's position relative to its leader's position. This estimated position is used by the *Average Positioning Mechanism*, *Out-Of-Zone Detecting Mechanism* and the *Out-Of-Range Detecting Mechanism* to calculate the robot's average position over time. It is also used to detect when the robot leaves the safe communication range or zones, respectively.

Lastly, the *Navigation Decision Mechanism* is used to make navigation decisions based on the number (or amount) of times that a robot's workers are out of range. More details pertaining to these behaviors and mechanisms will be presented in Chapter 2. The last point of interest with Figure 1.9 is in regards to the *Other "future" task-related connections.* This shows where additional behaviors (and their control mechanisms) would connect to the existing HCSMRS framework to provide more complex task-related behaviors that rely on the communication hierarchy. However, since the focus of this thesis was on the communication aspects of the framework, such additional behaviors were not implemented.

As will be seen in Chapter 2, all of the behaviors and mechanisms were implemented using (*Neuron Networks*) [2]. The entire system combines all behaviors to form a larger network in which sensory data enters the system, is processed, and a reactive steering decision is obtained. The system provides fast flow-related control representing a completely reactive system, making it quite robust. The choice of the neuron networks has additional advantages in that it lends itself well to the area of nano-technology since it reduces all processing to simple elementary interconnected

processing modules. Such modules are very similar to *gates* found in electronic circuitry. In fact, the entire neuron network control system developed here could be easily implemented on an electronic chip (see Figure 1.10). Basically, this chip hosts the HCSMRS-related networks as well as any other task-related networks, which are user-defined. The chip would take incoming electronic sensor signals and output actuator signals (e.g., motor steering commands). The benefit of this approach is that we obtain a fast reactive system that is shrinkable to very small robots. There is however, one disadvantage of this approach in terms of algorithm design. Implementing various behaviors, determining their interconnectivity/priorities and producing behaviors to perform complex tasks results in a significant amount of thinking and design. It is a similar analogy to that of the tediousness encountered when using a simplified programming language such as assembly or machine code.

1.3 Contributions

As will be discussed in the next section, there has been much work pertaining to multi-robot systems and their hierarchical communication strategies. Comparisons with our work will be made in that section. Here, however, for the sake of clarification, we list the contributions of our work.

The main contribution of our work is that we present a novel approach to designing a hierarchical communication strategy for a multi-robot system. Our design is based on fast, reactive neuron networks that represent the necessary communication and control mechanisms to achieve communication and coordination among the groups of robots. A secondary contribution is that we provides an implementation of these networks in JAVA as well as performed various tests to show how the system behaves. A third contribution is that we provide a brief analysis of the system performance for

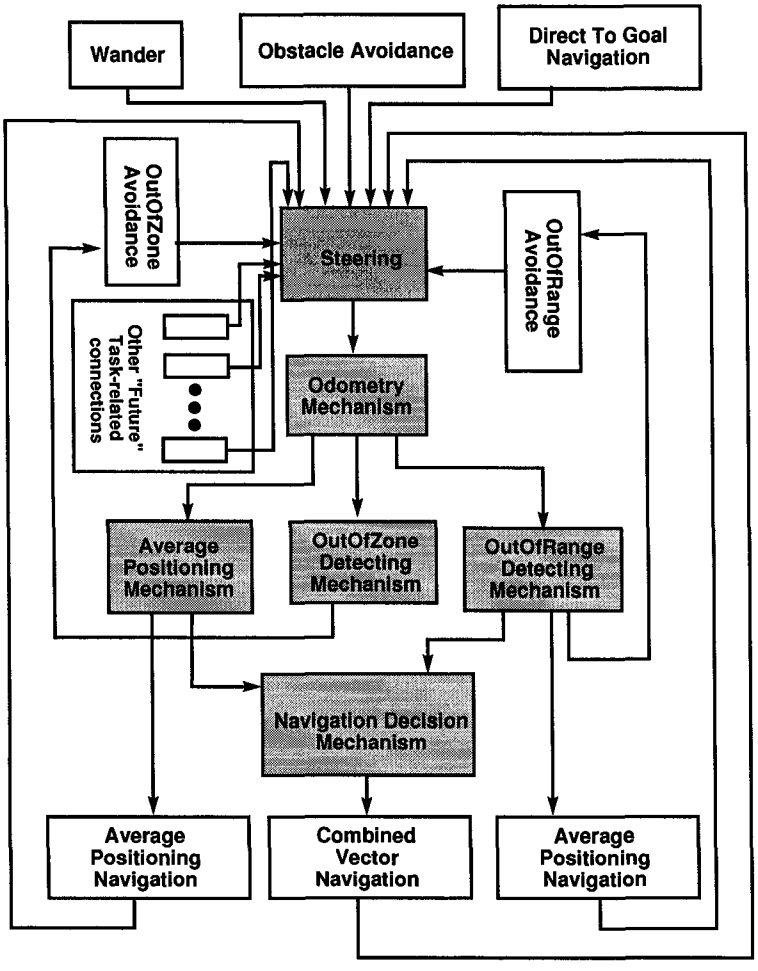


Figure 1.9: Framework of HCSMRS-related behaviors and mechanisms.

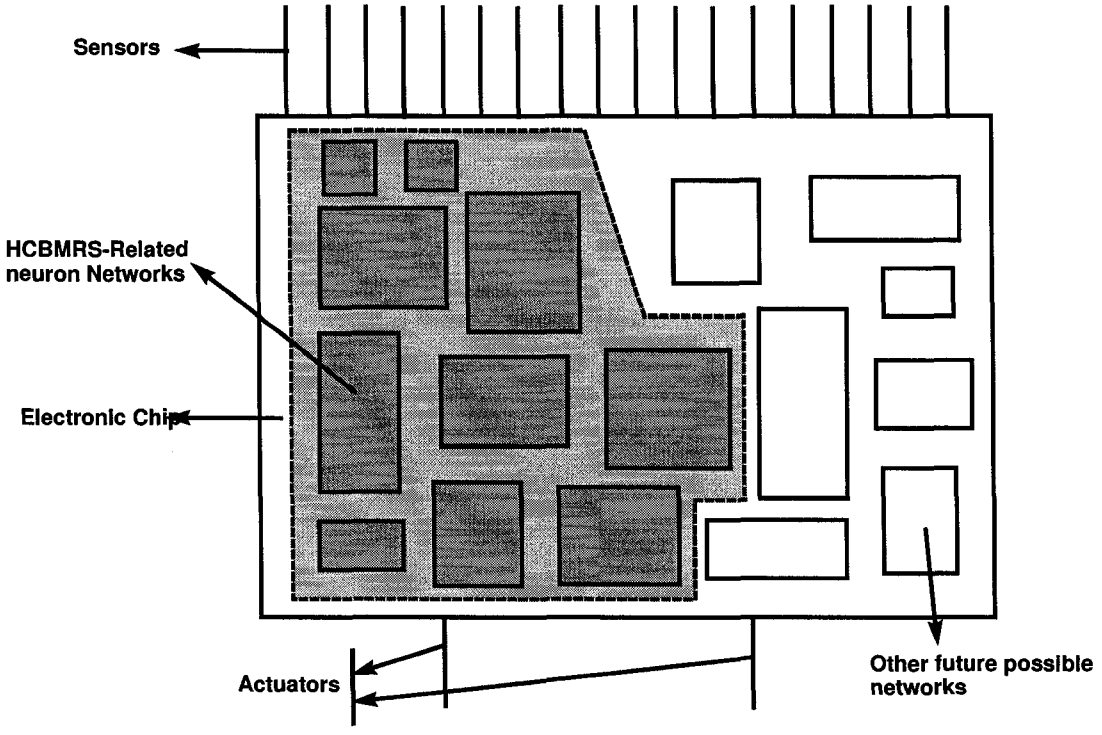


Figure 1.10: Concept of hardwired neuron networks embedded into an electric chip. Each block of electric circuit embedded expresses a behavior or a mechanism. The connection of these blocks shares the same structure with the connection in Figure 1.9

a two-level hierarchy in terms of environmental coverage showing its benefits over a non-hierarchical approach.

1.4 Previous Work

Recently, the use of hierarchical communication structures for multi-robot systems has grown considerably. Most of the previous work pertaining to communication architectures has focused either on (1) self-organization and formation control or on

(2) investigating policies for dynamically choosing (or switching) the leader robot. Although there has been considerable work over the years pertaining to pattern formation and leader selection, we give only a very brief survey of the more applicable work here.

In regards to self-organization, Damoto et. al. [5] proposed a "Super Mechano Colony" (SMC) which is a hierarchical/distributed multi-robot system composed of communication groups that follow a two layer "parent-children" relationship. In their work, "children" robots can be transformed and operated as independent agents when required. Thus, the colony can change its shape as needed at any time. Noreils [20] presented a self-organized hierarchical communication-based multi-robot system consisting of local supervisors and their "executers". The leadership is pre-determined according to the specific characteristics of the robots. The communication groups are organized by the supervisor through a protocol consisting of three mutual message exchanges. Fierro et. al. [10] uses the leader-following approach to achieve the path-following performed by a team of well-grouped mobile agents, which organized by the leader robots. The leader robot takes the role on planning the trajectory and assign the tasks to its worker robots. This is very similar to one of our path-concentrated coverage test scenario. As presented here, our work is different from [10] in that, instead of requiring the leader robot to plan the path, our system manually defines the paths, although further extensions to our work would likely overlap the research of [10].

Formation control is another attractive area of multi-robot systems. Much work has focussed on strategies for creating and maintaining a leader-following hierarchical formation of multiple robots. For example, Beard et. al. [1] presented a hierarchical control architecture to be used in the formation of spacecrafts. In this architecture,

all spacecrafts are centrally controlled by a supervisor through a central formation controller. Each of the spacecrafts are also equipped with a local controller, which is in charge of controlling its assigned spacecraft and communicating with the top formation controller. Also, Wang [26][27] developed several nearest-neighbor tracking strategies for navigating a fleet of robots in formation. In their strategies, beginning with the leader robot, all of the follower robots determine their motion status according to the tracking information from their nearest robot neighbor. By recursively tracking with the nearest neighbors, the formation can be created and maintained. Based on the work done in [26] and [27], Wang et. al. [28] applied the same navigation strategies to control the formation of moving rotational and translational micro-spacecrafts. By applying nearest neighbor tracking control rules, both the formation and the attitude of the micro-spacecraft is maintained. Yan et. al. [29] and Kapila et. al. [15] both applied a leader-following control strategy for satellite navigation so as to maintain formation in earth's orbit. In that article, the author presented a nonlinear dynamic method to determine the motion of followers relative to their leader's trajectory.

The hierarchical control structure is also applied to more specifically coordination-related applications. For example, Sugar and Kumar [22] developed a leader-worker architecture for a real time multi-robot system to control multiple robots while they are moving an object cooperatively. In this architecture, one leader robot takes the role of path-planning and communicating with its worker robots via sensor signals, while worker robots plan their own motion according to the coordination information received from the leader. Desai et. al. [6] applies a leader-worker architecture to control a group of robots so that they move within formation in a decentralized way. That is, when performing navigation, each worker robot coordinates with each other locally by using *feedback control laws*, such as the distance and orientation between

two worker robots or between one robot and an obstacle, but not relying on the information from the leader. A similar leader-worker coordination work was also investigated by [21]. In their work, the authors addressed a "highway" problem, where they adopt decentralized control laws to determine the velocity of the follower vehicles according to the coordinate information transmitted from the leader vehicle. Egerstedt and Hu [8] set a virtual leader to track the path. In the meantime, worker robots locally track the path according to the coordinates received from the leader and its neighbors. A virtual leader strategy is also used in [9] where, combined with artificial potentials between two neighbor robots, the authors set a moving reference point as the virtual leader to determine the best direction the worker robots should move towards.

In this previous formation work with respect to hierarchical systems, there are fairly strict requirements on keeping the worker robots in positions relative to both the leader robot and the neighbor robots. Our work does not address such specific formation control, but instead our zone partition mechanism allows the worker robots to move freely within their designated zones. In a sense, the worker robots do maintain relative formation relative to their leader. One difference in the work presented here is that maintaining formation within the HCSMRS depends only on the communication signals from the leader robots. This avoids the need for complex coordination and control rules among different robots.

In addition to the hierarchical arrangements, a recent area of focus has been to study the effects of setting up a leader-worker hierarchical architecture in multi-robot systems. For example, some work has focussed on the tradeoffs between the cost of a hierarchical architecture formation and the efficiency of the whole team's performance. Pertaining to this work, Tanner et.al. [24] defined the stability of a

Leader-to-Formation hierarchical communication architecture. They use this strategy to characterize how the motion of the group leader affect the motion of the whole group. Also, Egerstedt et. al. [13] and Ji et. al. [14] organized their robot team into a multi-layered structure. The authors analyzed the tradeoffs between the complexity of their hierarchical network and its performance and then tried to examine a compromise between the hierarchical and non-hierarchical structures. In this thesis, we too attempt to examine the effects that the leader's motion has on its workers and also show how our hierarchical structure compares to its non-hierarchical counterpart with respect to environmental coverage. Our HCSMRS also does not have any tradeoff issues with respect to the complexity of the hierarchy. That is, regardless of the number of layers, the coordination and communication behaviors and mechanisms remain the same since the entire design is naturally recursive. However, we did not attempt in this thesis to address the efficiency issues in regards to the number of layers in the hierarchy.

Recently, the discussion related to the stability of control formation for a hierarchical communication structure has become more popular. Tanner et. al. [24][25] defined and analyzed the *input-to-state* style stability of a leader-follower control formation. This stability is based on the leader inputs to the internal state. Combined with the feedback information among the worker robots, the stability is used to constrain the internal communication to achieve the stability of the whole formation. Furthermore, some work has focussed on the *abstraction of formation* to make the the task assigning of leader robot easier and more efficient. For example, Tabuada et. al. [23] discussed the feasibility of extracting a smaller control system that maintains its own group formation using graph theory. By doing this, the extracted system can be considered as a single entity that is involved in communication with the top-level leader. This abstraction largely simplifies the information transmission between different levels.

Work pertaining to leader selection has also been quite prevalent in the area of multi-robot systems. Miller [19] proposed a team of robots organized by a dynamically changing leader robot. Their research assumes that all robots can sense the beacons of all other robots. Robots are denied leadership if they sense a "higher" serial number encoded in beacons of other robots. Through comparison, the robot with the highest serial number is selected as the leader. Also, Dias et. al. [7] presented a market-based approach to coordinate multiple robots. In such a MRS, robots trade their own task allocation to maximize the "profits" and minimize the "cost". Through judging the "profit" made by each task allocation, the leader, which is the winner in the trade, can be selected.

Most of the previous work has focused only on the cooperative mechanism among worker robots or between worker robots and the group leader. However, Li et. al. [18] described a dynamic multi-layer hierarchical communication architecture and discussed the cooperation approach among different communication groups. In that article, authors proposed a three layer cooperative mechanism, combined with Markov processes and evolutionary computations to achieve the cooperation among different teams.

Although much of the previous work involved the setting up of a leader-follower hierarchical structure for organizing robots, the work focused more on task assignments, communication protocol and formation controls. In addition, the architectures are often dynamic, that is, each robot within the team has an opportunity to take on the role of the leader. Also, the number of robots within each group can change over time (i.e., children robots can join or leave one group according to environmental changes). For example, the work presented in [6] modeled the formation of a team

of robots while they are navigating terrains and developed the mechanism for transmitting formation information from one robot to another. In each transmission, both the group shape and the number of robots within a group may change.

Our HCSMRS presented in this thesis does not investigate these various dynamical hierarchy changes. Instead, our work is very similar to the robot team architecture presented in Li et. al.[18], wherein robots are organized into a multi-layer hierarchical structure. In both our work and that of [18], the communication among different groups within the same layer or different layers are via group leaders. The leader is fixed and pre-determined, playing the role of task allocation. The worker robots are in charge of executing the assigned tasks or distributing their own tasks to its sub-layer worker robots.

Our work differs from [18] in that we focus only on the mechanisms required for setting up the hierarchical communication architecture and basic group behaviors that are required for specific navigation patterns. We show the advantages that HCSMRS has over a non-hierarchical architecture through experimentation. Unlike [18], we do not focus on the cooperation and coordination aspects among different communication groups nor attempt to judge the *Quality* of the structure. Also, we do not investigate the task allocation and communication protocols among cooperative robots. Instead, we attempt to show how our robot team performs in terms of environmental coverage by using various navigation mechanisms that allow us to focus the team on particular areas of an environment.

Another aspect of our work that differs from previous work is in regards to how we implement the hierarchical control mechanisms. In contrast to all of the previous work mentioned, we implement our control structures as networks of neurons. As

mentioned in the previous section of this thesis, this allows the mechanisms to be reduced in size, hence facilitating the possibility of dramatically reducing in size the entire MRS. Also, this approach produces a reactive-based system with very fast response time. We followed the approach of Lanthier [17] and Lanthier et. al. [16] to implement these various behaviors and control mechanisms using networks of neurons.

1.5 The Robot Model

Before discussing the details of our communications strategy, it is necessary to describe the robot model with respect to the robot's capabilities. The robots in our HCSMRS are equipped with six antennae-like obstacle sensors, which are used to detect obstacles, two motors for driving the robot, a position/orientation sensor, an Out-Of-Range detector and an Out-Of-Zone detector. (See Figure 1.11).

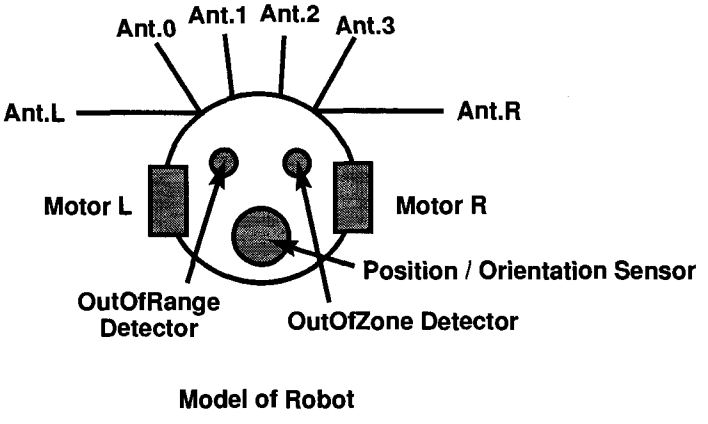


Figure 1.11: Our robot model showing its various sensors and actuators.

In the subsections to follow, we discussed how each of these logical sensors and detectors can be implemented on real robots.

1.5.1 Obstacle Sensors

Obstacle sensors are used to detect the presence of various objects in the environment so as to allow the robot to move along a collision-free path. There are many types of sensors that can be used for detecting the proximity of obstacles. The simplest approach is to use tactile sensors (e.g., bumpers or whiskers) since they are easy to interface to the hardware and are less prone to outside noise. Some designers prefer to use infrared (IR) proximity sensors due to the fact that they are non-mechanical and are less prone to physical damage. In larger robots, the trend is to use IR range sensors or sonar sensors that provide an estimate of distance to the obstacles. These tend to be more difficult to interface but do allow the robot to slow down when approaching obstacles. However, they are prone to error and can be quite coarse in their ability to detect fine obstacle features. In our robot model, we decided to represent the obstacle sensors as simple tactile whiskers (or antennae), but in a real robot, other more sophisticated sensors could be used. We modeled each antennae as an angled line segment coming from the robot. We assume that an obstacle is detected whenever the outermost tip of the antenna lies within an obstacle boundary. As a result, our robot detects an obstacle only when it is very close to it and constantly emits a binary signal indicating the presence or absence of an obstacle at the location of its outermost tip.

1.5.2 Position/Orientation Sensor

In order for a robot to navigate efficiently and ensure that it remains with communication range, it must keep track of its location in the environment. One of the most basic navigation elements required to determine a robot's current position and orientation is that of a position and orientation sensor. There are two strategies in maintaining a robot's position, namely *absolute positioning* and *relative positioning.*

Absolute Positioning

Absolute positioning is based on the idea that some external source provides positioning information with respect to some common coordinate system. Hence at any time, a robot using absolute positioning would be able to determine its location based on current data, regardless of its previous motion. Perhaps the most commonly used position estimation mechanism is that of a GPS (Global Positioning System) which determines a robot's absolute position by receiving signals from multiple satellites and then calculating its position. GPS systems are accurate to within a few meters and are mainly used for outdoor scenarios. Due to some unavoidable circumstances, such as signal delay, obstructions that block or reflect signals, receiver clock errors, reduced satellite visibility, satellite geometry shading, etc, the information provided by a GPS is sometimes unavailable. In addition, GPS systems cannot be used reliably in an indoor setting.

Another common approach used by robots to localize themselves is based on landmark recognition. During the robot's travel, its position can be calculated with respect to its relative distance between various identifiable landmarks. To achieve positioning, landmarks must have a fixed and known absolute position. They must also be visible and distinguishable. The identification of landmarks is often difficult, especially when the landmarks are natural as opposed to man-made. Thus, a robot

using this strategy for positioning would require much computational power, possibly sophisticated sensors and perhaps manual installation of landmarks (i.e., markers) throughout its environment.

Relative Positioning

A more reliable strategy for determining a robot's position is to use relative positioning. With relative positioning, a robot maintains its position relative to some previously computed (possibly moving) location. Hence the origin of the coordinate system for the robot's current position, may vary over time. In our system, leader robots could track and monitor the workers in their group and communicate their position to them over a wireless link. This is a kind of *local* GPS system. In this case, the group robot positions would be always relative to the leader's position. The advantage of this strategy is that it need not rely on GPS availability or landmarks, since the leader could monitor using other techniques, and perhaps even through a local omnidirectional video camera. As the leader moves, it would need to inform all workers to update their coordinate system reference point, and hence their own estimated position.

In a more robust system, it is best to have multiple layers of information. In cases where the worker robots are not visible from their leader, there would be no way to monitor and update the robot's position. Most robots in existence will implement some form of *odometry* or *dead reckoning* technique in order ro estimate their own position independently. The principle behind odometry is that of measuring the wheel revolutions and compass angles to compute the offset (from some known starting position) as the robot moves. Due to inaccuracies of wheel encoders and compasses, all such position estimates will grow in error over time , and sometimes very quickly. Therefore, odometry data must be occasionally updated by re-establishing a reference

point at regular intervals. This can be done when the robot regains visibility with its leader or when a GPS device comes back online and is able to validate the robot's estimated position. In some cases the robot may need to travel back to their leader for recharging, and in this scenario, its position estimate is reset to the leader's position. We have selected the odometry approach as the technique used by our robot model for maintaining its position. It is the least expensive approach and can provide good short-term accuracy.

1.5.3 OutOfRange Detector

Our HCSMRS requires the robots to perform their tasks within a localized communication range. To maintain communication, robots must continuously detect whether or not they are out of communication range. The OutOfRange detector used in our robot is responsible for performing such detections.

One approach to detect an out of range state is by measuring the signal strength (e.g., wireless signal) between a robot and its group leader. If the signal strength drops below some threshold, the worker robot is assumed to be outside the range of reliable communication. In reality, the signal strength from the leader is non-uniform, in that it may vary at different locations. This is true even if the distance between the worker robot and the leader are the same. (see Figure 1.12).

A different approach for detecting range exiting is to simply examine its estimated location (e.g., coordinate found using GPS or odometry). This provides a more uniform detection range since the robot merely needs to ensure that it lies within a certain radius from the leader's center. In Figure 1.13, the distance D_1 or D_2 between a robot and its leader determines whether or not it is within the safe communication

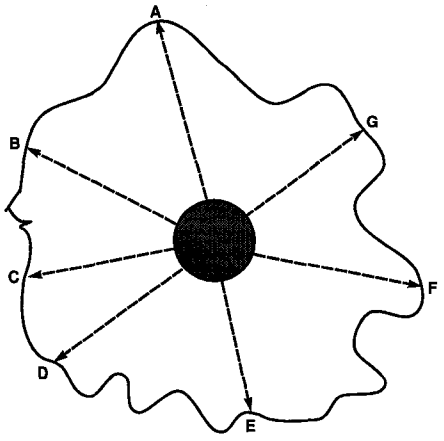


Figure 1.12: The non-uniform communication range.

range below threshold distance R. We chose to model our robot using this type of OutOfRange detection since we readily have available the robot's position estimate and did not need to worry about estimating signal warping throughout the environment. It is a simplified model and ignores the effects that obstacles may have on the communication signal strength.

1.5.4 OutOfZone Detector

For some applications, the communication range of a HCSMRS leader may be partitioned into several zones, and a number of workers may be assigned into each zone. We discussed such partitioning in the next section. Worker robots are assigned to a zone and must remain in that zone to perform their task. On occasion, a robot may stray from its zone and must take steps to regain contact with its designated zone. This can be done through leader-monitoring or through the robot's own estimation of its position. The actual implementation of this detector would depend on the type of zone chosen. For our model, we make use of each individual robot's odometry

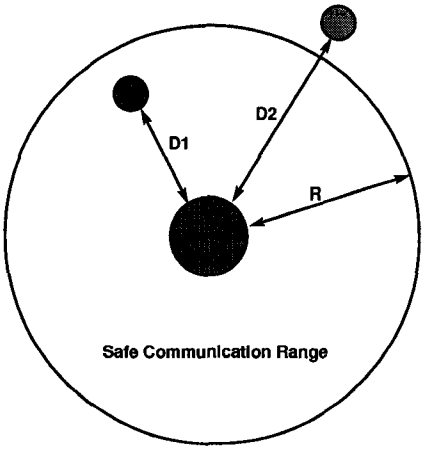


Figure 1.13: A uniform safe communication range radius R around a leader robot.

readings to detect when the robot leaves its designated zone.

1.6 Hierarchical Communication Model

In the previous section we discussed our robot model. In this section we discuss the model of hierarchical communication that is used in our system, which makes use of the robot model just described. We begin by discussing the communication range and zone partitions and then the overall communication hierarchy.

1.6.1 Communication Range

Recall Figure 1.1B which shows a two layer hierarchical communication model. In that model, the worker robots form a communication group which has a local group leader. The worker robots are required to remain within a *safe* communication range defined by a circle centered around their leader. In addition to having a safe communication

range, we also define another range called the *alarming* range (see Figure 1.14), within which, worker robots are urgently required to return back (based on the OutOfRange detector) within the safe range to avoid loss of communications. D_1 should be set to the maximum distance from the leader such that communications remain reliable without becoming intermittent or interrupted. D_2 should be chosen as the range in which communications are still available, but perhaps somewhat intermittent.

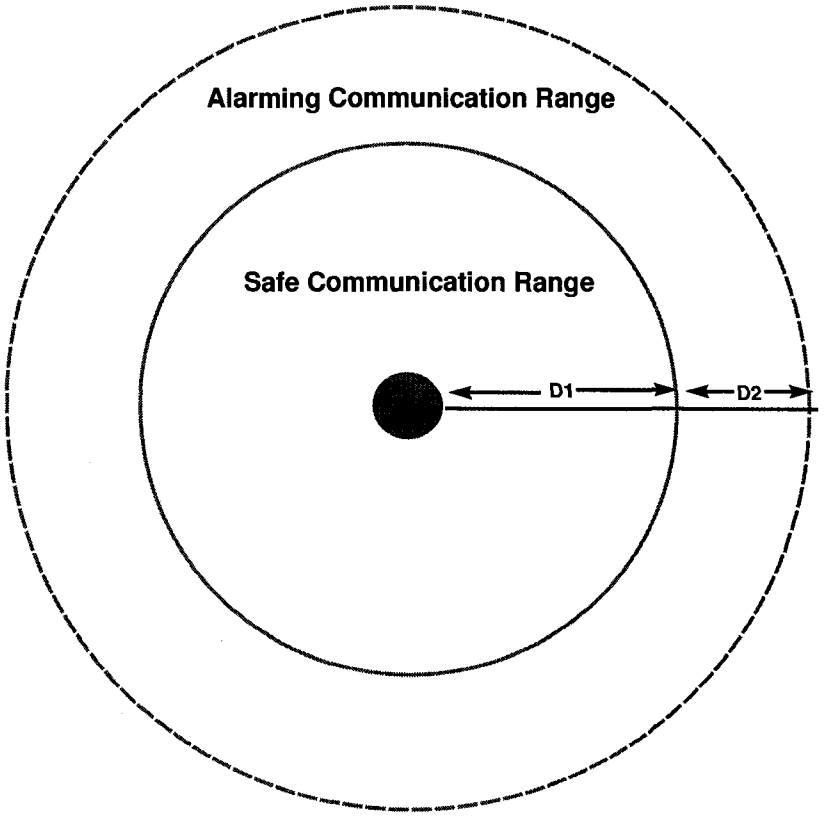


Figure 1.14: The communication range model showing safe and alarming communication ranges.

1.6.2 Zone Partition

Since worker robots within the same group may operate in close proximity, they may spend much of their time and energy avoiding one another. By partitioning the whole communication range into several zones and assigning a certain number of worker robots to each zone, such clustering can be avoided. This would allow individual workers to concentrate on their tasks at hand more fully as well as reduce inter-robot sensor interference. Thus, the whole efficiency of the HCSMRS can be greatly enhanced.

There are many ways in which the individual zones can be demarcated. Three such partitions are shown in Figure 1.15. In Figure 1.15A, each of the four robots are restricted to unique quadrants with respect to the leader's location. This simplifies computations, as will be seen later, since the trigonometric functions are straightforward along the X and Y axis. The partition of Figure 1.15B allows an arbitrary number of zones which is more flexible in regards to the number of robots in each group. However, the trigonometric functions are slightly more complicated and the worker robots have less work area and are more likely to overlap into other zones. The partition in Figure 1.15C reduces the chance of robot collisions, but does not assign a uniform work load to each robot. The four zone partition (Figure 1.15A) is used by our robots in this thesis.

Before executing any tasks, a number of worker robots are assigned to each zone and they are instructed to remain in their zone. Whenever a robot leaves its designated zone, a mechanism is triggered, based on the OutOfZone detector, to cause the robot to head back within its zone.

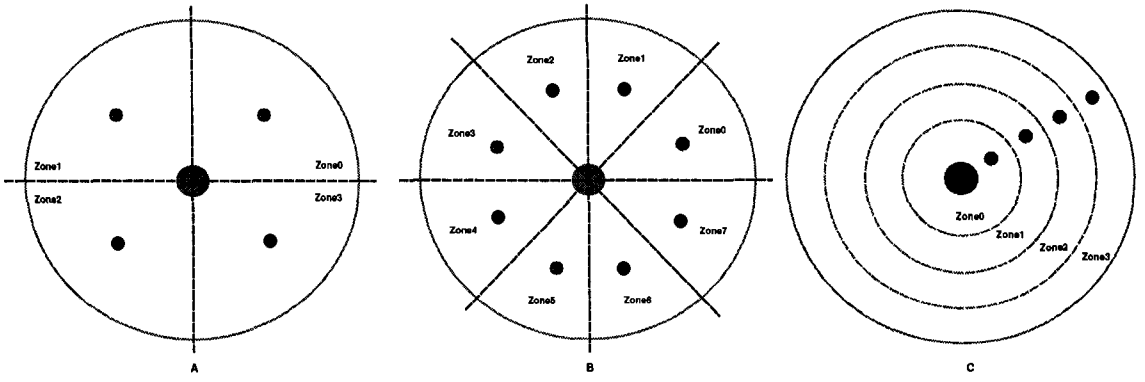


Figure 1.15: A: Four zone partitions. B: Eight zone partitions. C: Concentric ring like partitions.

Since the worker robots cannot receive reliable signals from the group leader when they are in the alarming communication range, the zone partitions become unusable there. Thus, if a worker robot is within the alarming range, and also outside its own zone, a higher priority is given to cause the robot to return to the safe communication range before it considers turning back to its own zone.

1.6.3 Multi-layer Hierarchical Communication Model

The HCSMRS discussed in this thesis is characterized by the multi-layer hierarchical architecture consisting of several multi-layered local communication groups (see Figure 1.16).

The robots in this system maintain a four layer hierarchical communication architecture. With the exception of the top layer leader all robots perform under the supervision of their group leader which makes higher level decisions for their worker robots. The highest level (i.e., topmost in the hierarchy) leader is responsible for

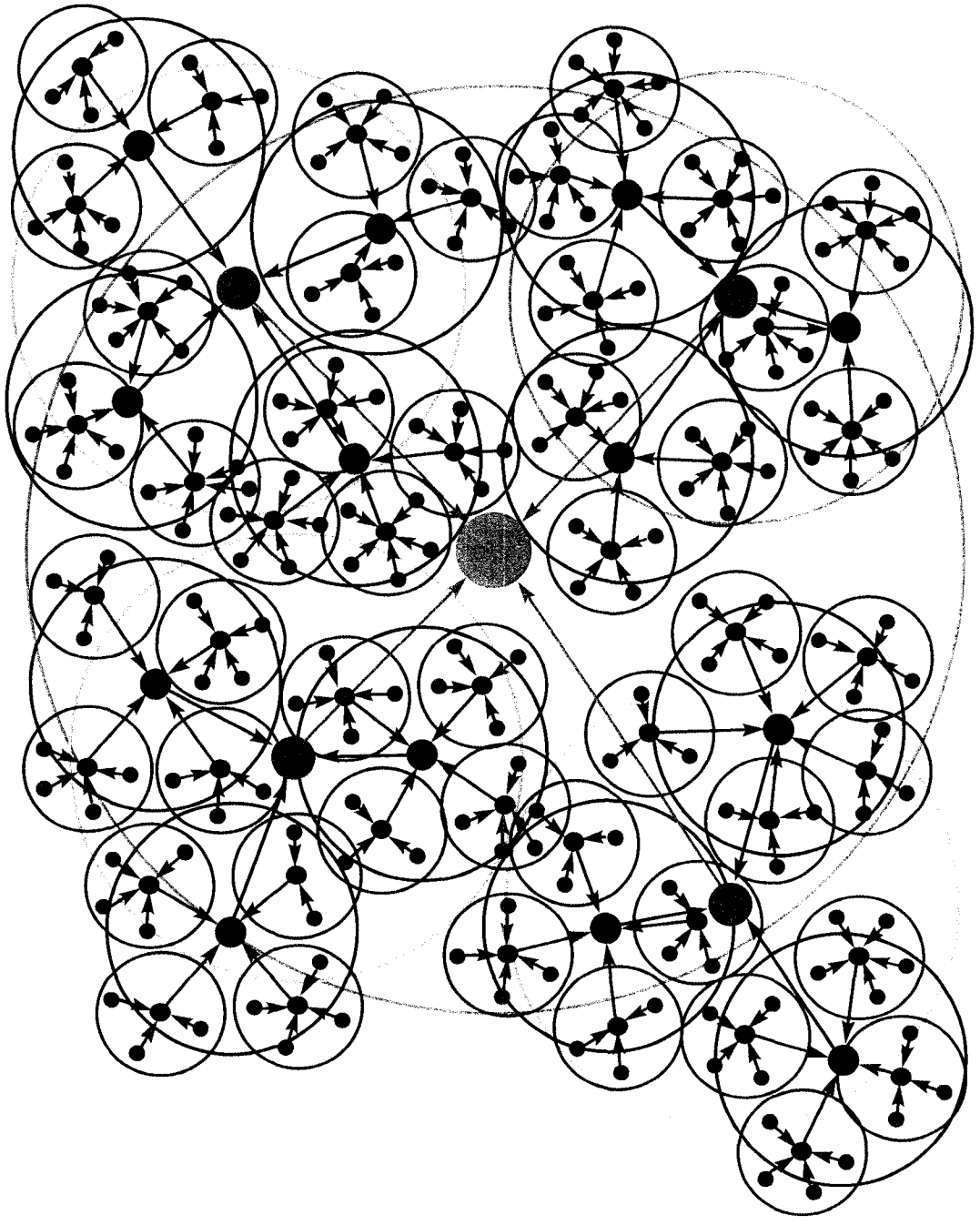


Figure 1.16: Four layer hierarchical communication model. At each layer, there are one or more communication groups, within each group, there is a group leader and several worker robots. The same-layer leaders are shown with the same color.

making general decisions pertaining to the whole multi-robot system. This highest level leader may be implemented in practice using a real person who monitors the overall task progress. The bottom layer worker robots always follow the instruction of their group leaders. Since every layer shares the same communication structure, the whole system is naturally recursive.

By using the multi-layered hierarchical communication architecture, robots maintain constant and reliable communication with their group leader. When compared to a centralized communication architecture which strongly relies on communication with a single base station, this system has greater potential for withstanding temporary communication outages. This allows allowing middle-level decision making to continue, even if the highest level decision-making knowledge is unavailable. Hence the HCSMRS is more fault-tolerant in terms of partial-communication reliability.

When compared to a decentralized communication architecture which requires robots to make their own decisions autonomously, this system incurs less of a computational burden on the individual robots since more difficult decisions are left to other robots up the chain of command. The HCSMRS's layered local communication architecture makes use of local leaders with better computing capabilities to analyze the environmental data and make the decisions for the local or entire multi-robot system.

1.7 Thesis Overview

The outline of this thesis is as follows: In Chapter 2, we present the basic behaviors used for controlling robots and creating the communication groups. In Chapter 3,

we explained the navigation-related behaviors and their implementations. Groups of navigation experiments are discussed, also in that chapter, the mechanism to implement each experiment is presented. Followed by the test result analysis in Chapter 4, where the corresponding test results are displayed in specific ways, by comparing with the test result made by distributed MRS, the advantage of HCSMRS is demonstrated. Chapter 5, the conclusion was drawn and our future work is presented and predicted.

Chapter 2

Robot Behaviors and Control Mechanisms

2.1 Neuron Networks

All behaviors and control mechanisms were implemented using *neuron network* which are composed of elementary interconnected processing modules, called *neurons.* These are similar to electronic circuit gates, which take incoming signals from other neurons and output transformed signals calculated through a transformation function. For example, Figure 2.1 shows a neuron model having $n + 1$ incoming signals I_0, I_1, ..., I_n, each with a unique weight W_i, where $0 \leq i \leq n$.

As shown in Figure 2.1, an activation value is calculated using the Equation 2.1.

$$activation = \sum_{i=0}^{n} I_i \times W_i \tag{2.1}$$

The function f, where $Output = f(activation)$, is used to generate a transformed

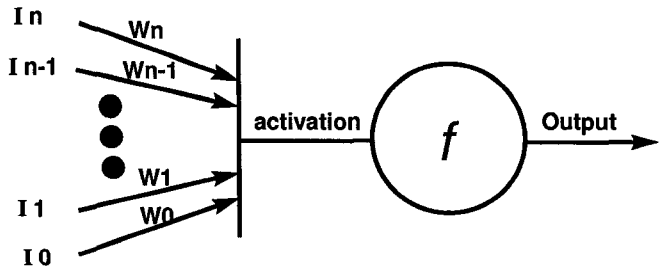


Figure 2.1: Neuron model.

output. These transformation functions give each neuron the ability to implement a certain mathematical calculation, such as basic operations for adding and subtracting signals. More complex operations can also be implemented through combining different types of neurons. In this thesis, we use neuron models as shown in Figure 2.2 to implement the HCSMRS-related behaviors and mechanisms. These are described as follows:

- **Standard Neuron:** Standard neurons are regular neurons as shown in Figure 2.1. The output of a standard neuron is equal to the activation value.

- **Binary Neuron:** The function applied in a binary neuron can be explained by the following equation:

$$Output = \begin{cases} 1 & if\ activation > 0 \\ 0 & if\ activation \leq 0 \end{cases}$$

- **Threshold Neuron:** A threshold neuron is a binary neuron with an adjustable cut-off value (i.e., threshold). If not otherwise specified, the threshold for all neurons is set to 1 by default. The output is determined as follows:

$$Output = \begin{cases} 1 & if\ activation \geq threshold \\ 0 & if\ activation < threshold \end{cases}$$

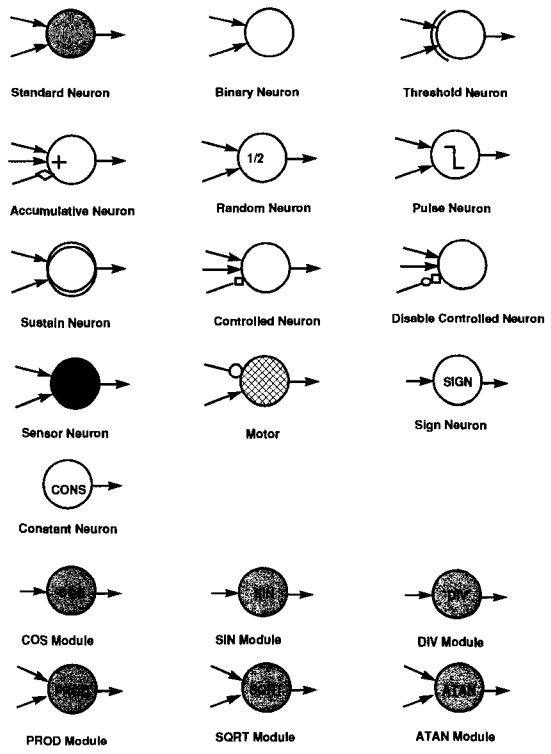


Figure 2.2: Neuron types.

- **Accumulative Neuron:** An accumulative neuron continually accumulates (i.e., using simple summation) the activation values until reset. This kind of neuron can be used as a counter. The output at any moment, is the accumulated value. When reset, the output and activation become 0.

- **Random Neuron:** A random neuron sends out a high signal with a random probability which can be specified, otherwise a 0 is the output.

- **Pulse Neuron:** A pulse neuron sends out a high signal when incoming signals drop down from high to low or rising up from low to high (i.e., detects rising-edge or falling-edge of signal). These neurons can be used as switches to enable/disable or reset other neurons when triggered.

- **Sustain Neuron:** Once activated with a positive activation value, a sustain neuron constantly outputs a 1 until its activation value becomes negative, then it outputs a 0.
- **Controlled Neuron:** A controlled neuron outputs its activation value, once it is enabled by a high signal coming in through a special control input link.
- **Disable controlled neuron:** A disable controlled neuron is inhibited once it is enabled by a high signal coming in through a disable control link.
- **Constant Neuron:** A constant neuron stores a fixed energy and always outputs this energy, which is usually 1 or -1.
- **Sensor Neuron:** Sensor neurons obtain the incoming signals from their related sensors and output this value.
- **Motor Neuron:** A motor neuron computes its output as a Standard neuron and then sends its output to its related actuator, which steers the robot to make a corresponding movement.
- **Sign Neuron:** A sign neuron computes its output as the sign of its activation value (i.e., -1, 0 or 1).

Neurons are used to implement basic operations. More complex math functions, such as "COS", "SIN", are implemented using *Modules*. Such modules can be created using electronic gates and signals, and in some cases, a network of fixed neurons hardwired (e.g., used as a look-up table). In this thesis, five kinds of modules are adopted.

- **COS Module:** The output of a COS module is calculated as $output = COS(activation)$.
- **SIN Module:** The ouput of a SIN module is calculated as $output = SIN(activation)$.

- **ATAN Module:** The output of an ATAN module is calculated as $output = ATAN(activation)$.
- **DIV Module:** The output of a DIV module is calculated as $1/activation$.
- **PROD Module:** The output of a PROD module is the product of its incoming signals.
- **SQRT Module:** The output of a SQRT module is the square root of its activation.

As described in Chapter 1, each behavior and mechanism is represented as a neuron network, which is composed of a number of interconnected neurons and modules (as shown in Figure 2.3). These neuron networks are interconnected together like high level "modules" and can be hardwired onto an electric chip as shown in Figure 1.10.

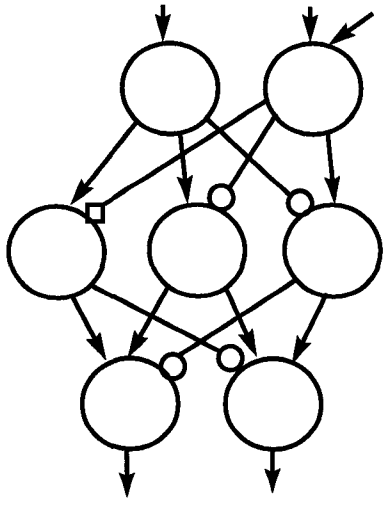


Figure 2.3: Neuron network model.

In this thesis, five types of links (see Figure 2.4) are used to connect neurons and modules.

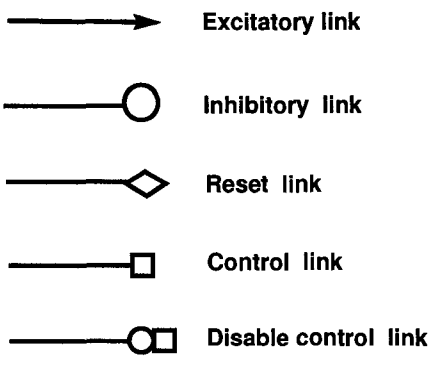


Figure 2.4: Link model.

- **Excitatory link:** output of a neuron directly to the input of another neuron.
- **Inhibitory link:** negated output of a neuron directly to the input of another neuron.
- **Reset link:** neuron to reset its activation to 0 when a high signal is sent along the link.
- **Control link:** connected to a controlled neuron. When a high (i.e., positive) signal comes along this line, the activation value of the control neuron is enabled (i.e., output = activation value). When a low signal is on this link, the control neuron is essentially disabled (i.e., output = zero).
- **Disable Control link:** to a controlled neuron to enable or disable its output. It works the opposite of a control link in that the control neuron is disabled as long as a high signal is on this link.

2.2 Basic Behaviors and Mechanisms For All Robots

All robots in a HCSMRS must be able to execute general movement, including *Steering*, *Wander*, *Obstacle Avoidance* and *Odometry*. This subsection will explain in detail the basic behaviors and mechanisms common to all robots.

2.2.1 Steering

The *Steering* network controls the overall steering of the robot. It consists of three binary neurons **Left**, **Forward** and **Right**, which connect to the **Left Motor** and **Right Motor** neurons as shown in Figure 2.5.

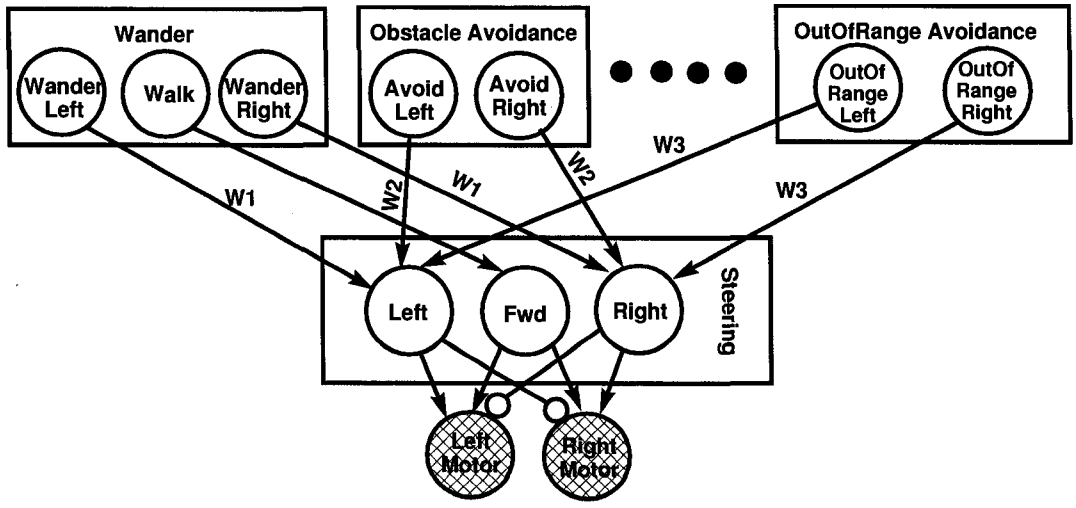


Figure 2.5: Steering network.

All networks (corresponding to the various behaviors and mechanisms of the robot) connect to the steering network via the **Left**, **Right** and **Fwd** neurons. That is, each network (i.e., behavior) requests the steering network to turn the robot in the direction

desired for that particular behavior. In a sense, the behavior networks all "compete" for overall control of the robot. Each connection made to the steering network uses a fixed weight, which indicates the priority of the behavior. The **Left**, **Right** and **Fwd** neurons simply sum these priorities and the highest priority "wins" control of the robot. This only works if the weights assigned to the links are unique powers of 2 (e.g., 1, 2, 4, 8, 16, etc..).

2.2.2 Wander

A wandering behavior allows the robots to make occasional random turns. Wandering behavior is fundamental to allow a degree of randomness in our system that helps the robot to escape difficult situations. Our wander network is based on the one used by RABI [17] and is shown in Figure 2.6. This network consists of three random neurons used to control when to turn, which direction to turn and how long to turn for.

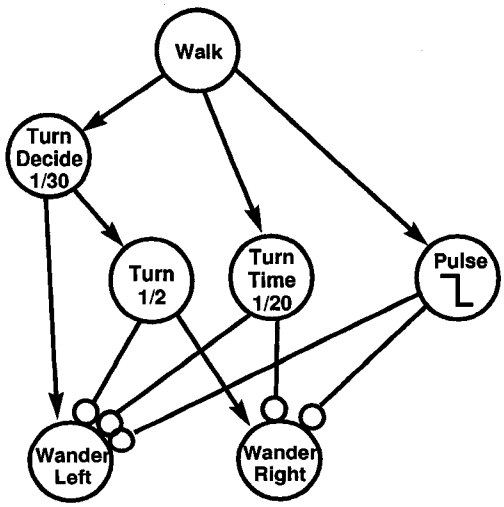


Figure 2.6: Wander network.

The wander network starts with a **Walk** neuron. When **Walk** is disabled, the **Pulse** neuron will send a high signal to disable the turning. The **TurnDecide** neuron randomly determines when to start turning. The **Turn** neuron randomly determines the turning direction and the **TurnTime** neuron randomly decides how long the turning will last for. Neurons **WanderRight** and **WanderLeft** are directly connected to neurons **Right** and **Left** in the *Steering* network. Each of the three random neurons is assigned a random value to represent the probability of exciting the neuron. Different rates set for each random neuron can result in different running traces. For example, the trace generated from robots having rate 1/20 for random neurons **TurningTime** and **TurningDecide** (see Figure 2.7(a)) is more linear than that generated from robots having rate 1/2 for those neurons (see Figure 2.7(b)).

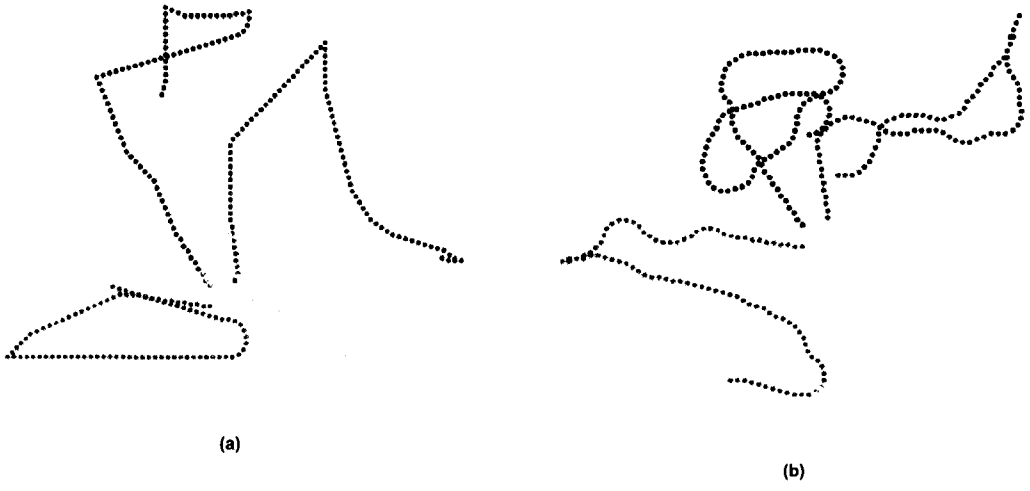


Figure 2.7: The traces of four robots with (a) rate = 1/20 and (b)rate = 1/2.

2.2.3 Collision Avoidance

To avoid colliding with other robots or the environmental boundary and obstacles, each robot incorporates *Collision Avoidance* behavior. Out network is based on the *Collision Avoidance* network of [17] and is shown in Figure 2.8.

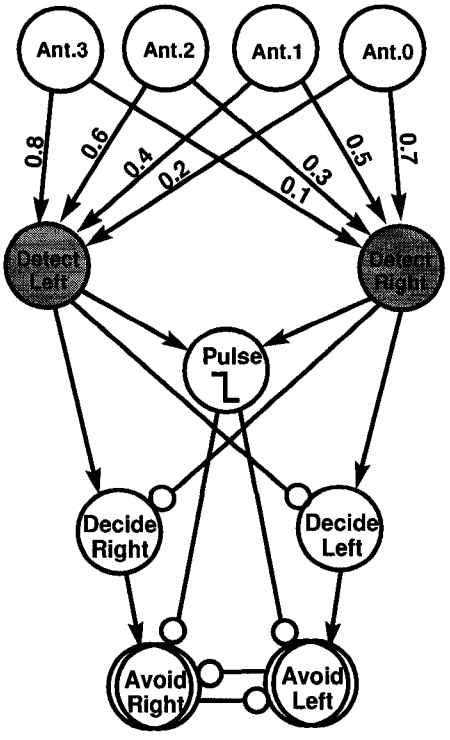


Figure 2.8: Collision avoidance network.

This network first receives the data obtained from the four front obstacle sensors. The two standard neurons **DetectLeft** and **DetectRight** determine the amount of sensor detection on each side of the robot. Each sensor connection to **DetectLeft** or **DetectRight** is given an unique weight. A larger weight indicates a stronger detection for that side of the robot. The **DecideRight** and **DecideLeft** compare

the total detection weights and make a decision as to which direction is best to turn in order to avoid the obstacle. The weights corresponding to the left side of the network are slightly larger than those on the right. This ensures that they do not cancel out one another. As a result, this network favors right turns in situations where the sensor neurons detect symmetrical shaped obstacles. Sustain neurons **AvoidRight** and **AvoidLeft** keep each HCSMRS robot turning towards one side until the collision status has been disabled. This eliminates the toggling back-and-forth that could occur in a corner situation. The falling-edge pulse neuron emits a high signal to disable the steering neurons once its sensors no longer detect a collision.

2.2.4 Odometry

The *Odometry* mechanism helps robots to locate themselves by keeping track of their position relative to some starting coordinate. The *Odometry* mechanism used in our HCSMRS is a self-updating neuron network, within which, a robot can calculate its current position from its previous position and direction.

To begin with, we make the assumption that robots do not turn while moving. Also, we assume that robots turn a fixed number of degrees (i.e., $\pm TurnUnit$) and move forward in small increments (i.e., +ForwardUnit). Both the **TurnUnit** and **ForwardUnit** are adjustable parameters. (see Figure 2.9).

Suppose that a robot starts from $P_1 = (X_{old}, Y_{old})$ with initial direction $D_1 = 0$. After turning left one unit and moving forward one unit, it arrives at point $P_2 = (X_{new}, Y_{new})$ where the robot's direction is now $D_2 = \theta$. Equation 2.2 and Equation 2.3 show how to calculate the new position.

$$X_{new} = X_{old} + ForwardUnit \times COS(\theta) \tag{2.2}$$

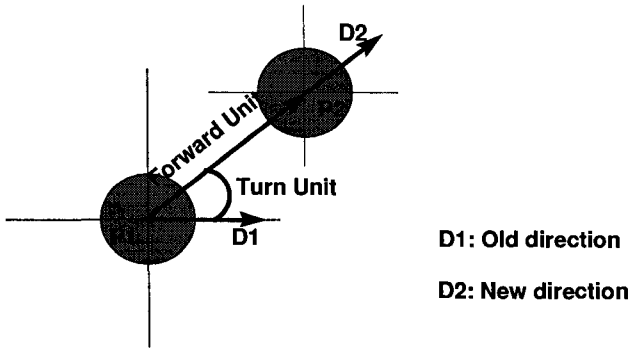


Figure 2.9: Odometry model.

$$Y_{new} = Y_{old} + ForwardUnit \times SIN\,(\theta) \tag{2.3}$$

Here, θ is the new direction which can be calculated as follows:

$$\theta = \begin{cases} (D_1 + TurnUnit) \div 360 & \text{if turning left} \\ (D_1 - TurnUnit) \div 360 & \text{if turning right} \end{cases}$$

The neuron network used to calculate the new direction is shown in Figure 2.10.

Two binary neurons **LTurn** and **RTurn**, connect to **Left** and **Right** from the *Steering* network, respectively. When the robot turns, **LeftD** or **RightD** receive a fixed activation signal with the amount **TurnUnit**. This signal is combined with the previous direction supplied by the **OldD** neuron. Since either **Left** or **Right** can be activated, but not both, the **NewD** neuron will receive the correct updated direction from either **LeftD** or **RightD**. We have omitted the details in our network regarding the MOD 360 calculation.

Once the new direction θ has been obtained, the new position can be determined according to Equation 2.2 and 2.3. Figure 2.11 shows the neuron circuit for computing

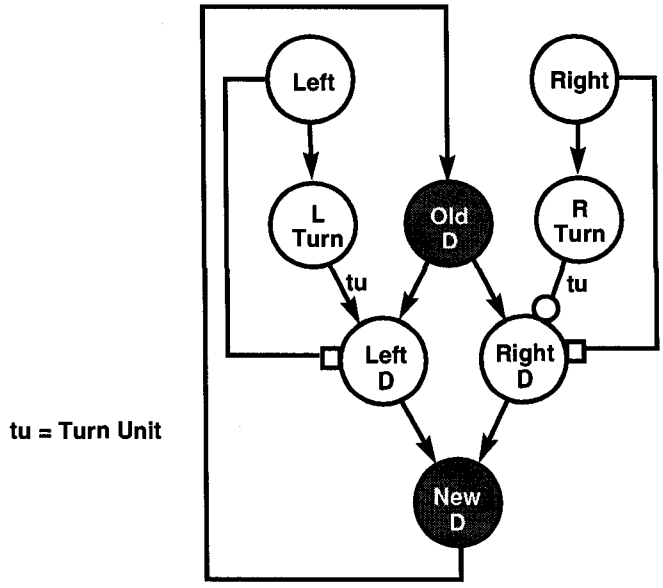


Figure 2.10: Portion of Odometry network used to calculate new direction.

this new position.

In this figure, neuron **NewD** comes from Figure 2.10 and is fed into the **COS** and **SIN** modules, through which $COS(\theta)$ and $SIN(\theta)$ are obtained. Neurons **Mag*COS** and **Mag*SIN** receive an excitatory signal from **COS** and **SIN** with a weight equal to *ForwardUnit* and a control signal from **FWD**, which comes from the *Steering* network and is enabled whenever the robot is moving forward. It is used here to enable the calculated value of $ForwardUnit \times COS(\theta)$ or $ForwardUnit \times SIN(\theta)$. These are added to the **OldX** and **OldY** neurons to obtain the new updated **NewX** and **NewY** positions, respectively.

The **InitialX** and **InitialY** neurons are used to supply starting coordinates for the robot when the robot leaves the leader (i.e, the worker is dispatched physically from the leader robot). The **AwayFromLeader** neuron is a sensor neuron that is

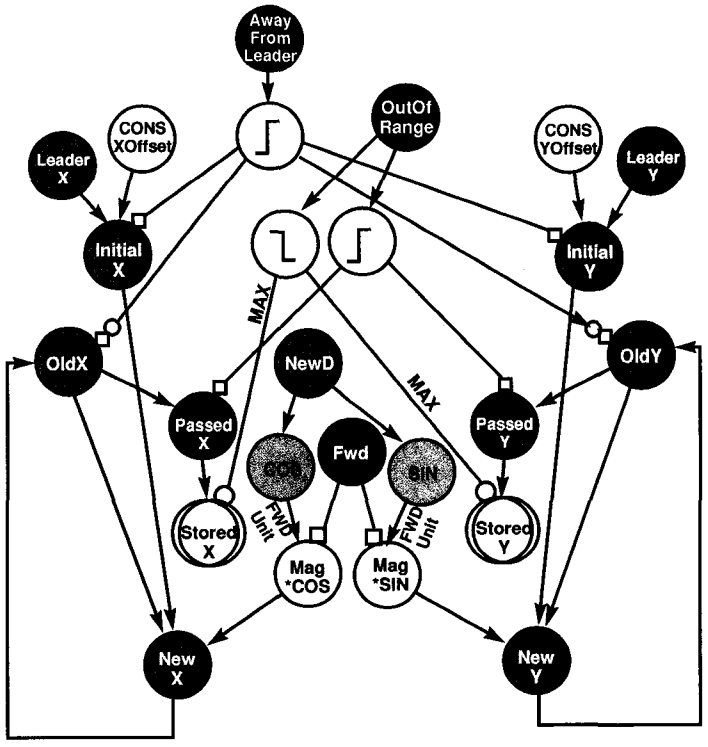


Figure 2.11: Portion of Odometry network used to calculate new position.

enabled when the robot leaves the leader robot. The rising edge pulse neuron sends a one-time control signal to the **InitialX** and **InitialY** neurons to enable this initial coordinate, while disabling the previous **OldX** and **OldY** neurons so that the old coordinate is not taken into account. The **InitialX** and **InitialY** neurons obtain their starting coordinate as the leader's coordinate plus some offset which corresponds to the offset from the robot that the worker robot starts moving on is own (e.g., perhaps a couple of feet away from the leader after being dispatched).

The network contains additional circuitry that stores the location of the robot in the **StoredX** and **StoredY** neurons whenever the robot goes out of range. As will be seen later, this coordinate is necessary in order to direct the robot back to proper

communication range. These stored values are reset when the robot is back in range again. The value of MAX in the network represents the largest negative number allowed in the system, so as to disable **StoredX** and **StoredY** when robots are out of safe range.

2.3 Communication-Related Behaviors and Mechanisms

Basic behaviors and mechanisms give a worker robot the most fundamental ability to walk safely and self-locate correctly. However, in order to keep the robot within communication range and within its designated work area, additional HCSMRS-related behaviors and mechanisms are necessary, such as an *OutOfRange Detection* mechanism (Section 2.3.1), an *OutOfZone Detection* mechanism (Section 2.3.2), an *OutOfRange Avoidance* behavior (Section 2.3.3) and an *OutOfZone Avoidance* behavior (Section 2.3.4). These are described in the sections to follow.

2.3.1 OutOfRange Detection Mechanism

To maintain a multi-layer hierarchical communication architecture, a robot must have the ability to monitor if it is out of the alarming/safe ranges or not by monitoring its distance from the leader. This can be done by continually comparing the distance with the communication range radius as shown in Figure 1.14. The distance between the robot and its leader is calculated from Equation 2.4.

$$Distance = \sqrt{(X_{new} - X_{new_{Leader}})^2 + (Y_{new} - Y_{new_{Leader}})^2} \tag{2.4}$$

Here, X_{new} and Y_{new} denote the current position of a robot and $X_{new_{Leader}}$ and $Y_{new_{Leader}}$ denote the current position of its leader. Based on Equation 2.4, the *OutOfRange detection* mechanism can be obtained, as shown in Figure 2.12, which makes use of the *Odometry* mechanism (Figure 2.11).

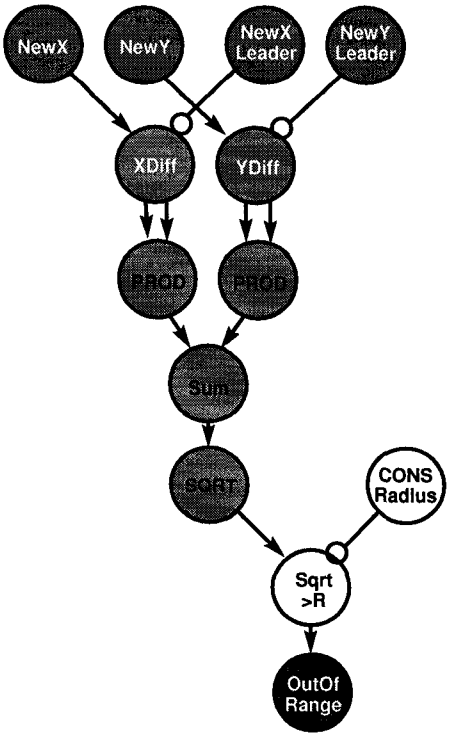


Figure 2.12: Out of range detection mechanism.

In the network shown in Figure 2.12, the **NewX** and **NewY** neurons provide the robot's current position from the Odometry network of Figure 2.11. Similarly, neurons **NewXLeader** and **NewYLeader** come from the corresponding Odometry network of the leader robot. Of course, this data would need to be transmitted via wireless communications from the leader on a regular basis as often as possible. Hence, we assume that sensor neurons are used to provide such input. This thesis

ignores details corresponding to how the data is transmitted and received. Neurons **XDiff** and **YDiff** individually receive excitatory signals from **NewX** and **NewY** and inhibitory signals from **XLeader** and **YLeader**, thus computing the difference in the position of the worker and the leader. The two **PROD** modules combined with the **SUM** and **SQRT** modules compute the distance between the robot and its leader. Binary neuron **Sqrt>R** is used to determine whether or not the *distance* is larger than the radius of the alarming communication range (as specified by the constant neuron **Radius**). Finally, the boolean result activates the **OutOfRange** sensor neuron, which later connects to the *OutOfRange Avoidance* network (see Section 2.3.3) to enable that behavior.

2.3.2 OutOfZone Detection Mechanism

For most applications, it is desirable to reduce the chance of collisions among robots. One way to do this is to assign work zones to individual robots. The distribution of unique non-overlapping work zones to individual robots has the added advantage of distributing the workload more evenly among the robots. The HCSMRS framework incorporates a 1-zone-per-robot approach in order to take advantage of more efficient task distribution.

In this thesis, we partition the leader's communication range into four zones (see Figure 2.13), corresponding to the Top Left (TL), Top Right (TR), Bottom Left (BL) and Bottom Right (BR) quadrants relative to the leader robot's last known location. Robots continuously monitor their location and if they detect that they move outside of their designated zone with respected to the last known leader location. They take steps to head back towards their designated work zone. The detection process has been implemented using an *OutOfZone Detection* mechanism.

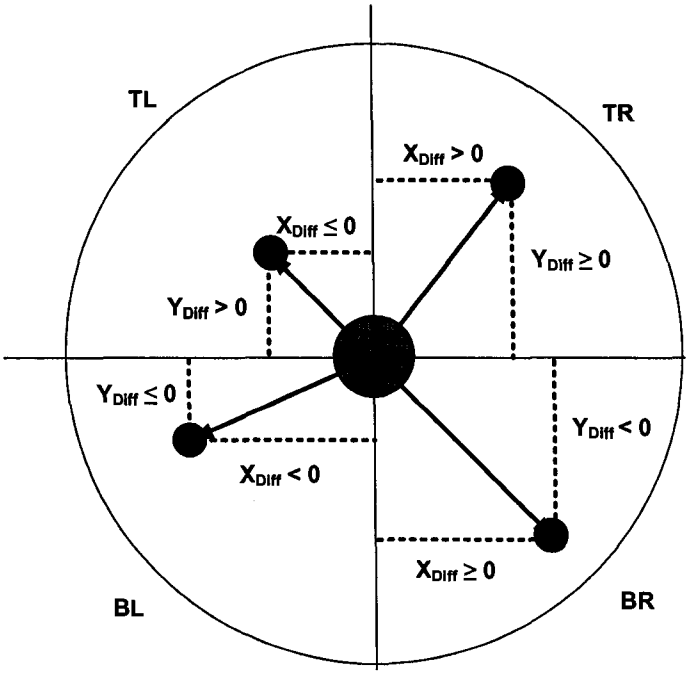


Figure 2.13: Four zones partitioned in the leader's communication range.

The worker robots determine their current zone by checking the sign of the difference between the X and Y values of its location with the leader's location. The values of X_{Diff} and Y_{Diff} are computed as shown in Equations 2.5 and 2.6.

$$X_{Diff} = X_{new} - X_{new_{Leader}} \tag{2.5}$$

$$Y_{Diff} = Y_{new} - Y_{new_{Leader}} \tag{2.6}$$

The robot then determines its zone by examining the signs of X_{Diff} and Y_{Diff} based on the following rules:

$$Zone = \begin{cases} TR & if\, X_{Diff} > 0, Y_{Diff} \geq 0 \\ TL & if\, X_{Diff} \leq 0, Y_{Diff} > 0 \\ BL & if\, X_{Diff} < 0, Y_{Diff} \leq 0 \\ BR & if\, X_{Diff} \geq 0, Y_{Diff} < 0 \end{cases}$$

Figure 2.14 shows the neuron network responsible for computing a robot's current zone based on these rules. The network is subdivided into two portions. The portion labeled PartA represents the calculation for the particular situation in which neither of the robot's X nor Y is zero, whereas the PartB portion calculates for the case in which either X or Y is zero.

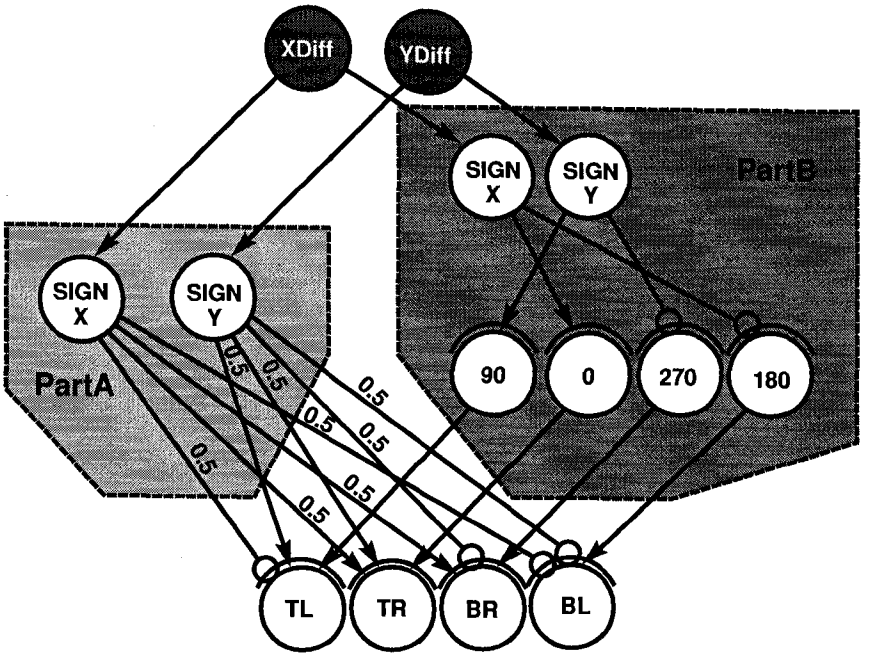


Figure 2.14: Network for computing a robot's zone.

In the PartA portion, neurons **XDiff** and **YDiff** are the neurons from the network in Figure 2.12. Two pairs of sign neurons **SIGN X** and **SIGN Y** examine the sign of the x and y differences, respectively. The left pair enables the appropriate zone neuron (i.e., either **TL**, **TR**, **BR** or **BL**) based on the quadrant that the robot lies in with respect to the leader location as origin. The right pair enables the appropriate degree neuron (i.e., either **0**, **90**, **180** or **270**) based on the boundary between zones

that the robot lies in with respect to the leader location as origin. Recall that the threshold neurons are enabled when a combined positive signal activation of 1 or more is obtained. In the PartB portion, the neurons **90**, **0**, **270** and **180** enable the appropriate zone according to which axis boundary the robot lies on with respect to its leader's x/y axis. For example, if the robot lies at coordinate $(x, y) = (0, 10)$, then **XDiff** has a value of 0 and **YDiff** has a positive value which causes **SIGN Y** to excite the **90** neuron. If the robot lies at coordinate $(x, y) = (0, -10)$, then **YDiff** has a negative value which causes **SIGN Y** to excite the **270** neuron because of the negative output of the SIGN neuron multiplied by the negative inhibitory link.

In addition to determining its zone, a robot needs to determine whether it is out of its specified zone. Each robot is assigned a zone upon startup. The robot then detects whether it is out of zone by comparing its assigned zone with its current zone. Figure 2.15 shows four such networks used for determining this "OutOfZone" status. A robot will use only one of these networks, depending on its designated zone. Each network makes use of three of the four zone neurons from Figure 2.14, namely **TL**, **TR**, **BR** or **BL**. The neurons **ID.TL**, **ID.TR**, **ID.BR** and **ID.BL** represent the zone to which the robot is designated (i.e., these are hard-coded for each robot). The **Out** neuron is a threshold neuron (with threshold of 1) that becomes enabled only when the robot enters one of the other 3 zones that do not correspond to its designated zone. This then enables the **OutOfZoneTR** sensor neuron which is used later to direct the robot back to its correct zone by using the *OutOfZone Avoidance* behavior.

2.3.3 OutOfRange Avoidance Behavior

Once a worker robot has determined that it is leaving the safe communication range and entering the alarming range (i.e., the **OutOfRange** neuron from Figure 2.12 is

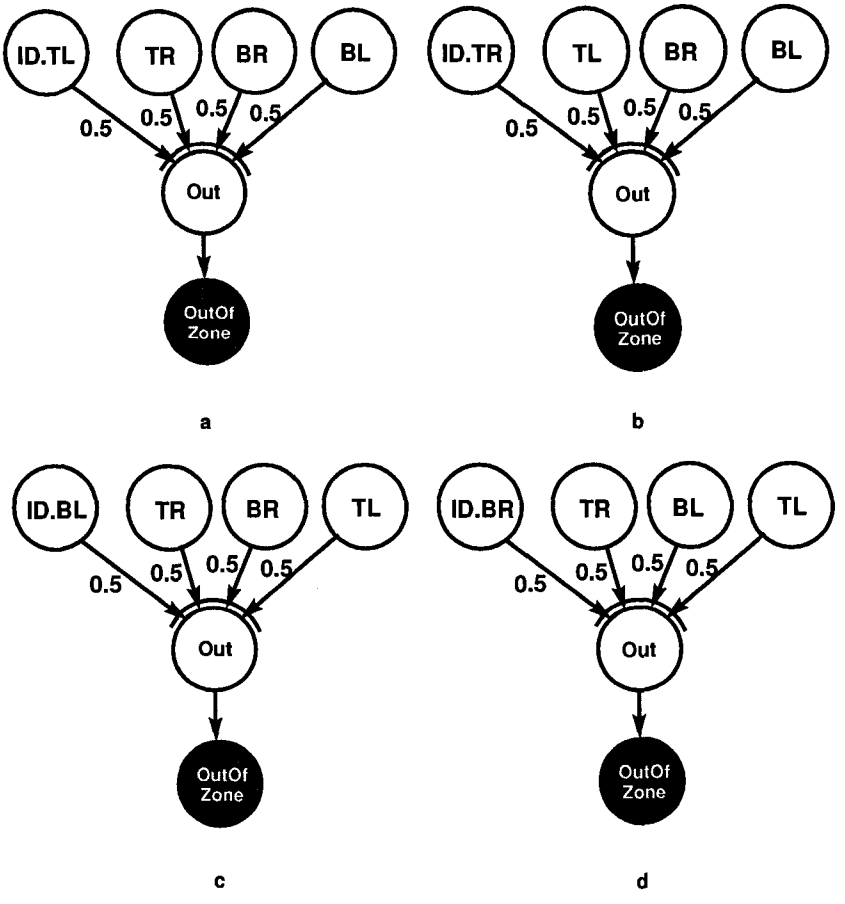


Figure 2.15: Four possible "OutOfZone" detection mechanisms.

excited), the *OutOfRange Avoidance* behavior kicks in. This behavior is intended to bring the robot back into safe communication range as soon as possible so that it does not lose contact with its leader. Recall from Figure 1.14 that communication with the leader is still possible within the alarming range, but may be less reliable.

This OutOfRange Avoidance behavior can be implemented in various ways, but we describe here one way which has shown to be effective at steering the robot back into the safe communication range.

If the leader robot remains stationary, then avoiding travel outside of the safe communication range is simple. Upon stepping into the alarming range, the worker robot merely needs to stop, turn around and head back towards its leader. However, since the leader is typically moving as well, complications arise in this strategy. First, the leader robot may move directly away from the worker robot and if the worker robot does not act quickly, the leader may soon be too far from the worker robot, causing the worker robot to become lost altogether. This problem can be overcome by allowing the leader robot to travel at a slower speed than its workers and by giving a high priority to the OutOfRange Avoidance behavior.

A second, more serious situation can occur when the robot is traveling alongside an obstacle as it enters the alarming communication range. The leader may move in a direction that situates the obstacle between the robot and its leader (e.g., move to the right in Figure 2.16). Hence, heading back directly towards the leader causes an obstacle collision and thus prevents the robot from achieving safe communication range again (see Figure 2.16).

Of course, the leader can always monitor its worker robots and intervene in order to help prevent them from leaving the safe communication range by communicating "turn back" messages. Our solution to this problem is to not head back directly towards the leader, but to head in a direction that is more likely not to get caught in a difficult obstacle collision situation. To do this, we consider the last known location of the worker that was within the safe communication range (called the last known "good" location). If the leader did not move, we can simply head towards this "good" location. If the leader does move, we should take this into account. As a compromise, we compute a combined vector heading for the worker robot. This heading vector is a combination of the vector towards the last known "good" location and the vector

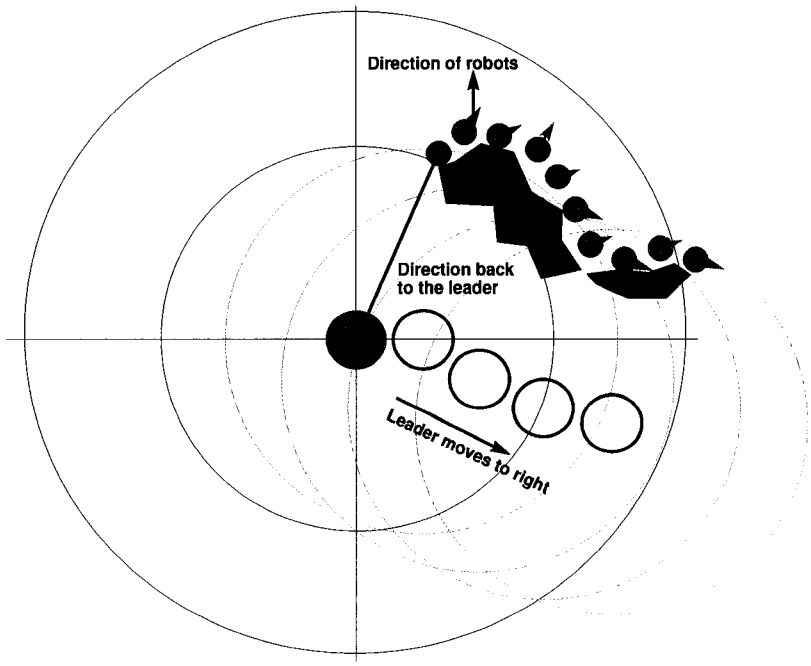


Figure 2.16: Situation that can cause a robot to delay getting back into safe communication range.

towards the leader's current position (recall that the leader still emits a wireless signal indicating its current location).

Figure 2.17 depicts the calculation of the combined vector V_C based on the vectors V_G (from the worker robot to the last "good" location) and V_L (from the worker robot to the leader's location). Here, the desired heading for the worker robot is the combined vector V_C which can be represented as a desired goal location of D. The worker robot need not determine the goal location D, instead, it simply needs to determine the direction of V_C. This will be called the *desired direction* and will be denoted as β.

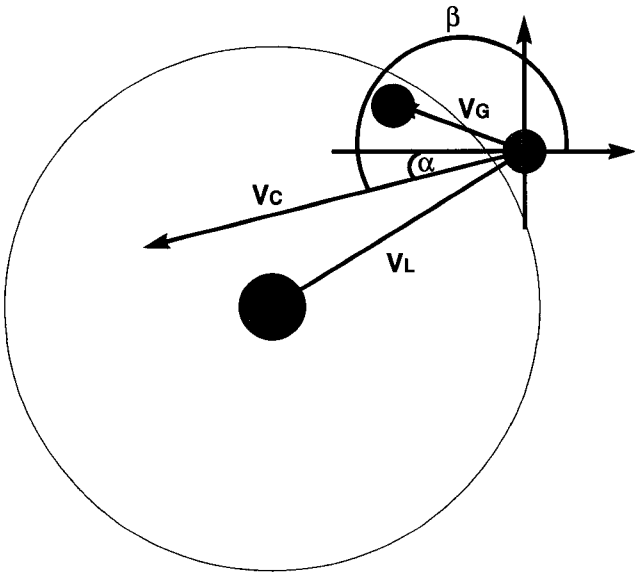


Figure 2.17: Calculation of the combined vector V_C.

Specifically, V_G is computed by determining the difference in X and Y coordinates of the last "good" location G and the worker's location W. Also, V_L is computed similarly using the location L of the leader and the worker W. These calculations are shown in Equations 2.7, 2.8, 2.9 and 2.10.

$$X_{DiffGW} = X_G - X_W \tag{2.7}$$

$$Y_{DiffGW} = Y_G - Y_W \tag{2.8}$$

$$X_{DiffLW} = X_L - X_W \tag{2.9}$$

$$Y_{DiffLW} = Y_L - Y_W \tag{2.10}$$

The combined vector V_C is then determined by first computing the values X_{Diff} and Y_{Diff} as shown in Equations 2.11 and 2.12:

$$X_{Diff} = X_{DiffGW} + X_{DiffLW} \tag{2.11}$$

$$Y_{Diff} = Y_{DiffGW} + Y_{DiffLW} \tag{2.12}$$

We can then compute the angle that V_C makes with the X-axis by computing the arc tangent of this difference by using X_{Diff} and Y_{Diff} as shown in Equation 2.13. We will denote this computed angle as α.

$$\alpha = arcTan\,(Y_{Diff} \times (1/X_{Diff})) \tag{2.13}$$

Figure 2.18 shows how α can be computed using a neuron network. In this network, neurons **NewX Leader** and **NewY Leader** correspond to the leader's location and are the same neurons as used in Figure 2.12. Similarly, neurons **NewX** and **NewY** represent the worker robot's location and correspond to the neurons with the same name from Figure 2.12. Neurons **XG** and **YG** represent the previous worker location and hence correspond to neurons **Stored X** and **Stored Y** in the odometry network of Figure 2.11. Modules **DIV** and **PROD** compose the template *Calculate* α, which will be reused many times in Chapter 3.

The output of neuron **ATAN** denotes the value of the angle between V_C and the X axis. In order determine the actual direction of V_C with respect to the positive horizontal axis (i.e., standard convention), we must examine eight situations corresponding to the the leader robot lying in one of the 4 quadrants (or on the axis) with respect to the worker robot's position. These 8 cases are depicted in Figure 2.19), which shows how the desired direction β of V_C is computed. As shown here, the sign of X_{Diff} and Y_{Diff} reflects the robot's current position relative to the leader. The combinations of the signs of these two values results in the eight situations shown. Recall that α is the angle that V_C makes with the horizontal axis of the coordinate system in which the worker robot's location is the origin. α is positive for the upper right and lower left quadrants since the ATAN function returns a positive angle in those quadrants. The eight cases shown in Figure 2.19 will be another template, namely *Eight Transferring Cases.*

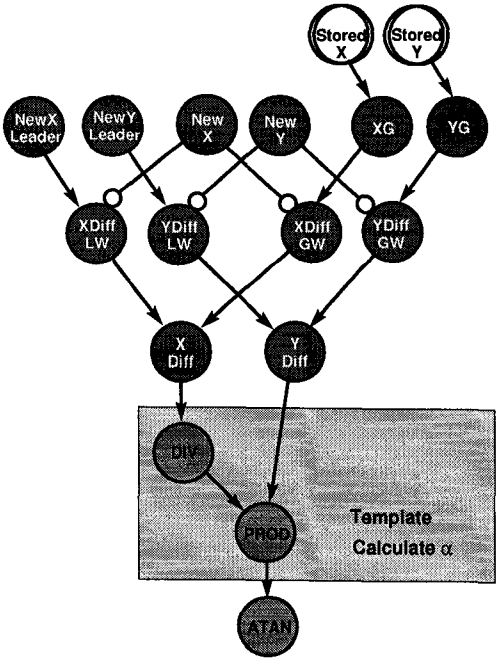


Figure 2.18: Network to compute the angle α.

Figure 2.20 shows the neuron network for computing β according to the eight situations in Figure 2.19. PartA implements the four situations corresponding to Figure 2.19 a, b, c and d.. The other four situations of Figure 2.19 e, f, g, h, are implemented as PartB of the network. Neurons **XDiff** and **YDiff** correspond to those of Figure 2.18. Sign neurons **SIGN X** and **SIGN Y** examine the sign of x and y differences. Four threshold neurons **TL**, **BL**, **BR** and **TR** are used to determine which situation the robot is in, corresponding to Figure 2.19 a, b, c and d. Note that the thresholds are set to 1, so these neurons all output zero in the axis-boundary cases. Neuron **ATAN** represents α and it comes from the network of Figure 2.18. Neurons **DTL**, **DBL**, **DBR** and control neuron **DTR** calculate the desired direction β for the four non-quadrant-boundary situations using the equations of Figure 2.19. Note that the **DTR** is a special case which simply enables the α value to be passed

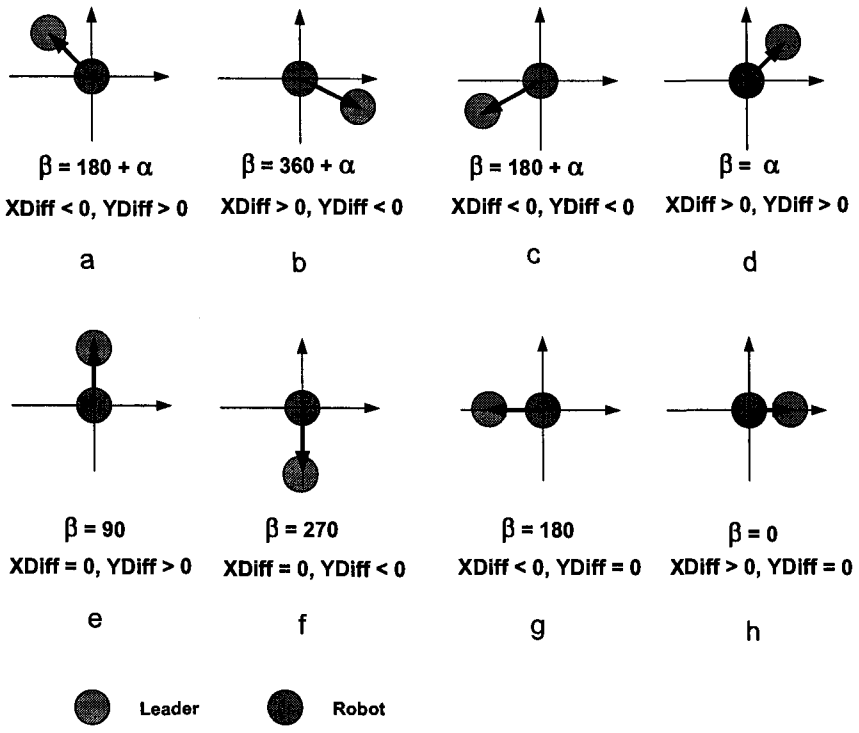


Figure 2.19: Eight situations for computing β from α.

through to **pTR** when the top right zone (i.e., **TR** neuron) is detected. The four control neurons **pTL**, **pBL**, **pBR** and **pTR** receive the excitatory signals from **DTL**, **DBL**, **DBR** and **DTR** respectively and the control signals from **TL1**, **BR1**, **BL1** and **TR1**. Since only one of **TL1**, **TR1**, **BR1** and **BL1** can be excited at any given time, only one of these four controlled neurons will be allowed to output the proper value of β which is indicated by the **DVC** neuron. This **DVC** neuron therefore outputs a value in the range of 0 to 359 to represent the degree orientation of the robot.

In PartB, the pair of sign neurons enables four degree threshold neurons **0**, **90**, **180** and **270** which represent the quadrant-boundary situations (i.e., e, f, g, h).

Since only one these four threshold neurons can be excited at any given time, only the matched degree will be obtained. Neuron **D** can perfectly disable the signals from PartB through receiving four negative controlled signal from PartA. This is for the non-quadrant-boundary situations, which allows only PartA to give signals.

Also, the most parts of this network can be considered as another template *Transfer to* β.

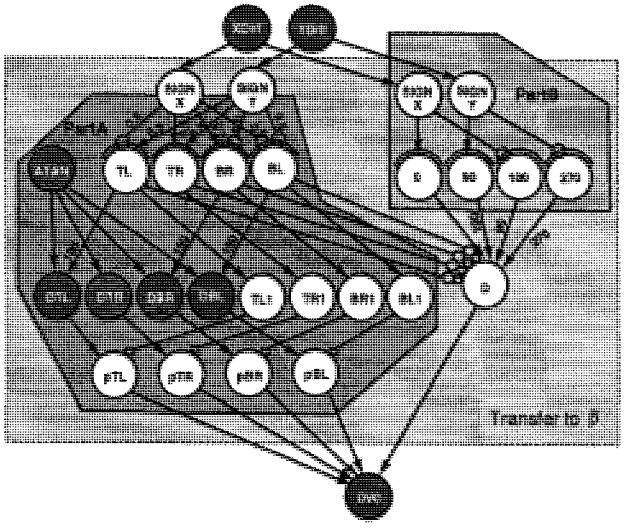


Figure 2.20: Network to compute the desired direction β of V_C.

Turning Back to the Safe Communication Range

At this point, the robot can use the network of Figure 2.20 to determine the direction in which it must travel in order to get back to the safe communication range. The robot must now have a means of steering towards that direction using some strategy. This can be done by comparing the absolute value of the difference between the robot's current direction and its desired direction. In order to reduce fluctuating left/right turns as it travels toward the desired direction, it is good to "roughly" aim the robot towards the desired direction. This is done by only turning

when the robot is significantly off from the desired direction. To do this, we can make use of the robot's pre-defined **Turn Unit** (i.e., refer back to Figure 2.9). If the absolute value of the direction difference is smaller than the **Turn Unit** the robot can continue moving forward, since it is going "roughly" in the right direction. Otherwise, the robot needs to adjust its course by turning towards the V_C direction.

Assuming that D_R denotes the robot's direction, and D_{VC} denotes the desired direction of the combined vector, this turning decision can be made as follows:

01. IF $(D_R > D_{VC})$ THEN
02. IF $((D_R - D_{VC} \leq 180)$ AND $(D_R - D_{VC} > TurnUnit))$ THEN
03. Turn Right
04. IF $((D_R - D_{VC} > 180)$ AND $(D_R - D_{VC} < (360 - TurnUnit)))$ THEN
05. Turn Left
06. ELSE
07. IF $((D_{VC} - D_R \leq 180)$ AND $(D_{VC} - D_R > TurnUnit))$ THEN
08. Turn Right
09. IF $((D_{VC} - D_R > 180)$ AND $(D_{VC} - D_R < (360 - TurnUnit)))$ THEN
10. Turn Left

This algorithm has been implemented in a neuron network as shown in Figure 2.21. Here, neuron **DVC** is from the network of Figure 2.20, which outputs the desired direction of the combined vector. The **NewD** neuron represents the robot's current direction and comes from the odometry network of Figure 2.10.

In this network, **DiffPos** is excited when $D_R - D_{CV} > 0$ and **DiffNeg** is excited when $D_{VC} - D_R > 0$. The constant neuron **TurnUnit** is the number of the degrees the robot turns each time it makes a turn decision. In our simulation, this is set to

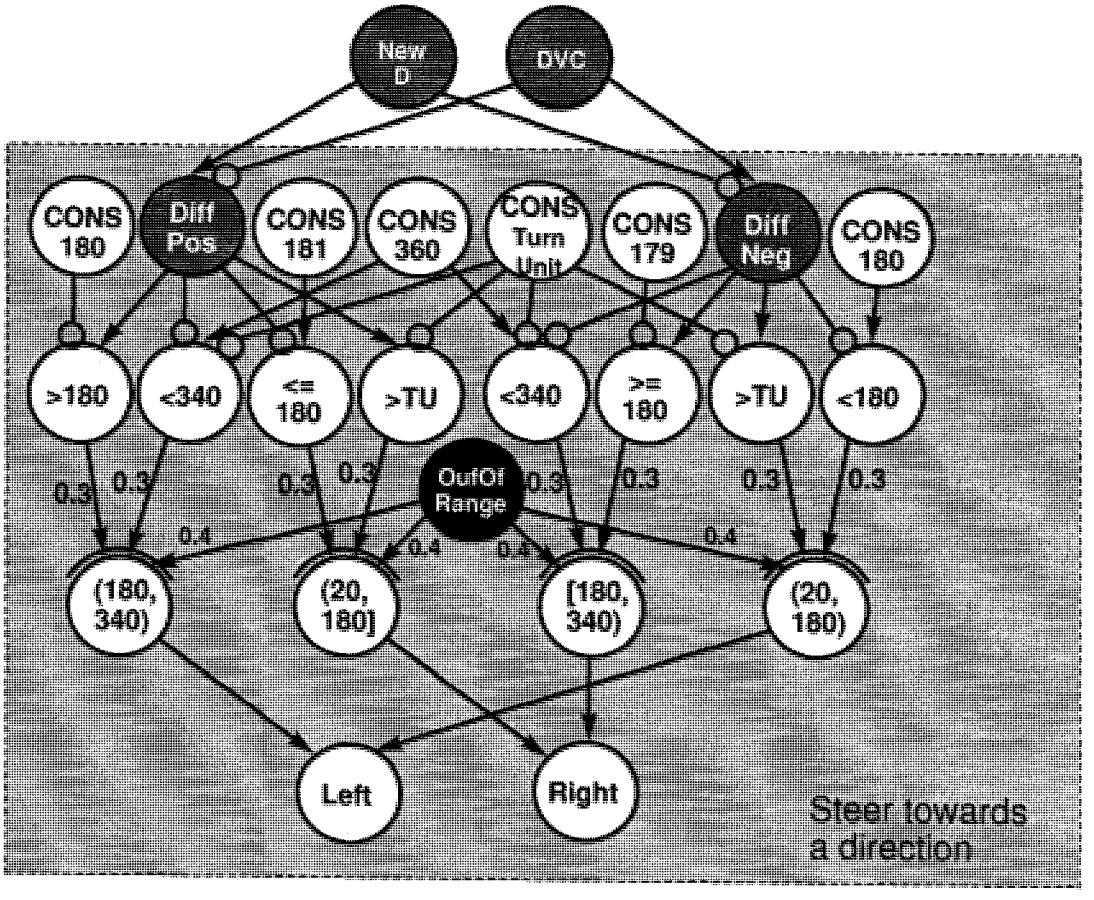


Figure 2.21: Network used to steer robot back into safe communication range.

20 degrees. The third row of neurons in the network compare various constant values (i.e., from constant neurons in the 2nd row) to determine whether the robot needs to make a turn which is more than 20 degrees or less than -20 degrees. The 4th row of neurons are threshold neurons that implement the IF statements of our algorithm. As a result, if the robot needs to make a turn more than the **TurnUnit** in order to head back into range, then either the **Left** or **Right** neuron is enabled. If the robot is already heading *close enough* towards the desired direction (i.e., the angle is less than the TurnUnit), then this network does not cause the robot to turn and the lower level

wandering behavior "kicks-in". As shown in this figure, this network can be used as a template *Steer towards a direction.*

In preceding discussion, we introduced four network templates, *Calculate* α, *Eight transferring cases, Transfer to* β and *Steer towards a direction.* These four templates compose a four step mechanism, specifically for those behaviors require robots to head to one or more target or steer towards an unknown direction.

Avoiding Obstacles

The above strategy performs well when there are no obstacles, because the worker robot is always able to quickly get back into safe communication range after a short excursion into the alarming communication range. However, when an obstacle is encountered during this process (as shown in Figure 2.22), the collision avoidance behavior may "kick-in" (since it has a higher priority) and causes the robot to be delayed in getting back into safe communication range. That is, there would be a conflict in regards to which direction to steer the robot. In the situation of Figure 2.22, the robot moves in direction $D2$ and while doing so, it leaves the safe communication range. In an attempt to head back into range, the *OutOfRange Avoidance* behavior is enabled and then the robot turns right. This suddenly causes a collision detection and the *Obstacle Avoidance* behavior kicks-in, turning the robot left again. Thus the robot can easily become caught in a situation involving alternating turns, resulting in a "stuck" behavior.

If the leader robot does not move, or moves very slowly, collisions with simple obstacles do not generally cause problems. For example, Figure 2.23 shows 2 examples in which the robot is able to reach the safe communication range after an excursion

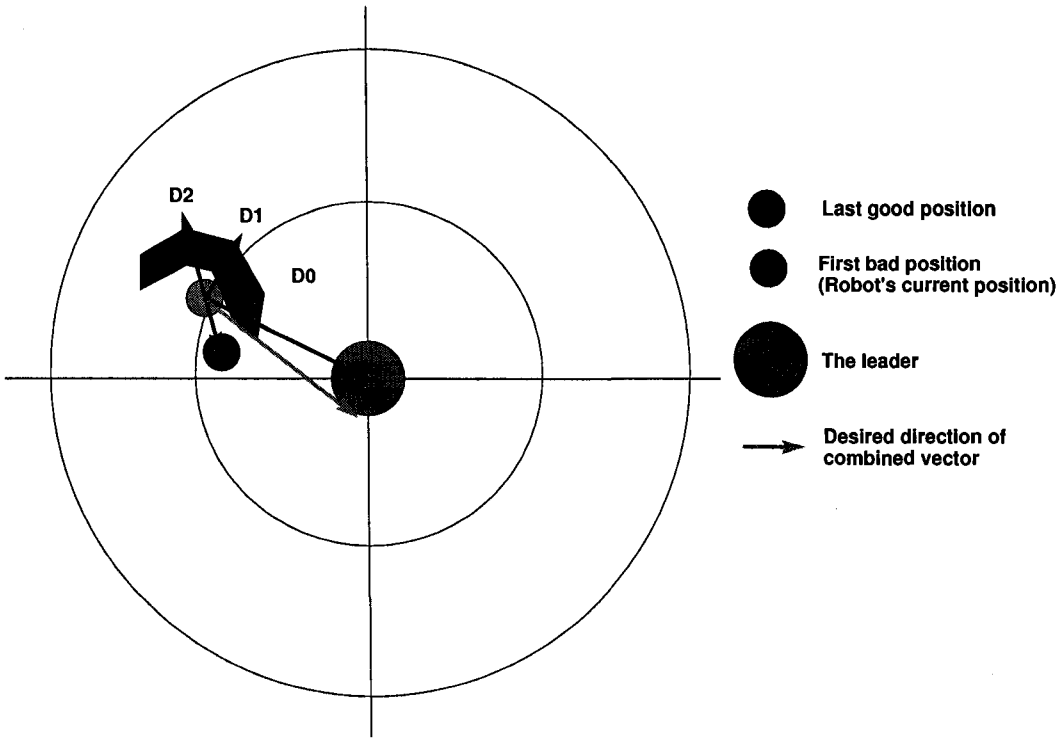


Figure 2.22: Situation causing the *OutOfRange Avoidance* behavior to fail.

into the alarming range. In Figure 2.23A, the robot travels along the obstacle until it reaches point $P1$. At this point, both *Obstacle Avoidance* behavior and *OutOfRange Avoidance* behaviors agree to turn the robot left. This agreement happens to lead the robot away from the obstacle, allowing the *OutOfRange Avoidance* behavior to complete unhindered. Figure 2.23B, shows a similar situation, in which the robot may actually travel all the way around the obstacle before successfully reaching the safe zone again.

However, the situation is not always this straight forward. Figure 2.24, for example, shows that if the obstacle is too large, then the robot may leave communication

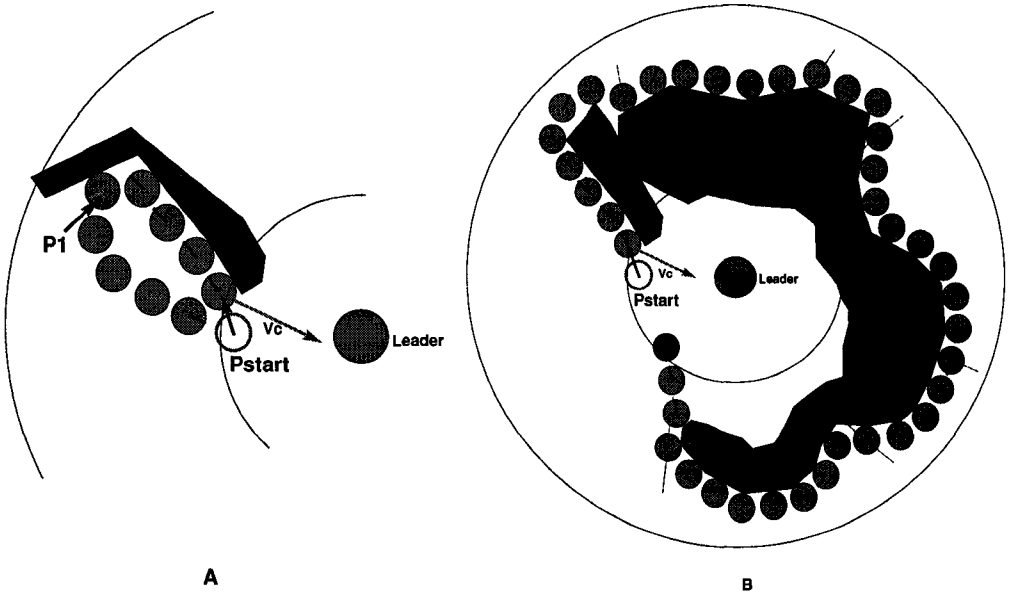


Figure 2.23: Examples showing how the *OutOfRange Avoidance* behavior successfully deals with obstacles.

range completely while performing obstacle avoidance before reaching the safe communication range. This could cause the robot to become completely lost if the leader is moving. In general, if the perimeter of the obstacles encountered is always less than the distance between the safe communication boundary and the out of range boundary. Then this problem is less likely to occur since the worker robot should have ample time to re-enter the safe communication range even if it has to traverse the entire obstacle.

Although we did not address this issue in the thesis, it can be addressed by adjusting the *Obstacle Avoidance* behavior. For example, if the robot detects an obstacle continuously on its right, it may choose to move left (i.e., away from the obstacle) altogether. This would require an adjustment to the *Obstacle Avoidance* behavior

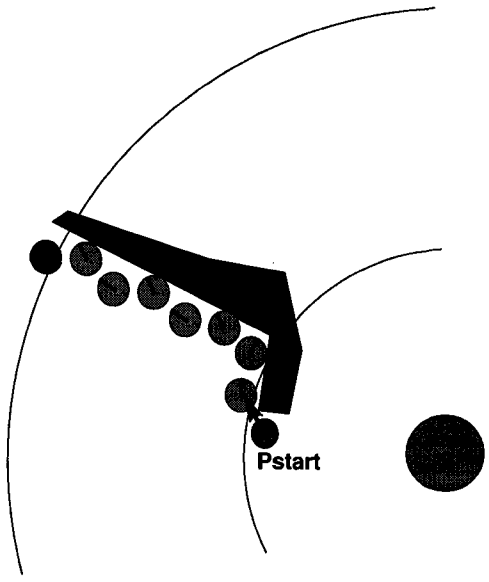


Figure 2.24: Examples showing how the *OutOfRange Avoidance* behavior can fail due to obstacles.

network.

Despite changes to the *Obstacle Avoidance* behavior, there can still be situations in which the robot may get "stuck". Figure2.25(a) shows an example in which the worker robot leaves the safe communication range and then has difficulty re-entering the safe range when the obstacle avoidance behavior kicks in. As a result, it ends up tracing the obstacle counter-clockwise. After reaching the other side of the obstacle, the robot tries to head towards the safe range again, but is blocked by the middle obstacle. As can be seen in Figure2.25(b), the robot then traces this obstacle and ends up looping clock-wise as it breaks free. This cycle may continue indefinitely as shown in Figure2.25(c).

A useful strategy that helps to escape such loops is to add a degree of randomness (hence unpredictability) in the robot's overall behavior. To do this, we enabled the

Wandering behavior to steer the robot "occasionally". That is, when the robot is not being turned by the *Obstacle Avoidance* behavior or the *OutOfRange Avoidance* behavior (i.e., when it is simply moving forward), the wandering behavior "kicks-in". This wandering will turn the robot slightly at random times (we used a probability of 0.25 for turning). While random movements do not necessarily progress towards a particular goal, they are quite successful at getting the robot "unstuck".

Considering again Figure2.25(c), a short straight line motion is noticeable as the robot heads back between the obstacles. In this short time, neither *Obstacle Avoidance* or *OutOfRange Avoidance* behaviors turn the robot, and this allows the lower priority *Wandering* behavior to have a chance at turning the robot and breaking the cycle. If the robot is turned left in this short period of time, then it will re-encounter the rightmost obstacle and allow the robot to travel clockwise around it to regain the safe communication range as shown in Figure2.25(d). Figure 2.26 shows a trace from our simulation depicting this exact scenario and how the robot was able to break free from the cycle.

2.3.4 OutOfZone Avoidance Behavior

The *OutOfZoneAvoidance* behavior is enabled when a worker robot has strayed from its designated zone. It attempts to steer the robot back into its own zone. While discussing this behavior, we assume that the robot is within a safe communication range at all times. We assume that the *OutOfRangeAvoidance* behavior has higher priority than the *OutOfZoneAvoidance* behavior. Here we discuss a solution for the 4-quadrant zone arrangement (other zone partitions are not addressed). Thus, when a robot leaves its designated zone, we assume that it is in one of the other three zones. Figure 2.27 shows the four possible situations for each of the four worker robots R_{TR}, R_{TL}, R_{BL} and R_{BR} which are designated to zones TR, TL, BL and BR,

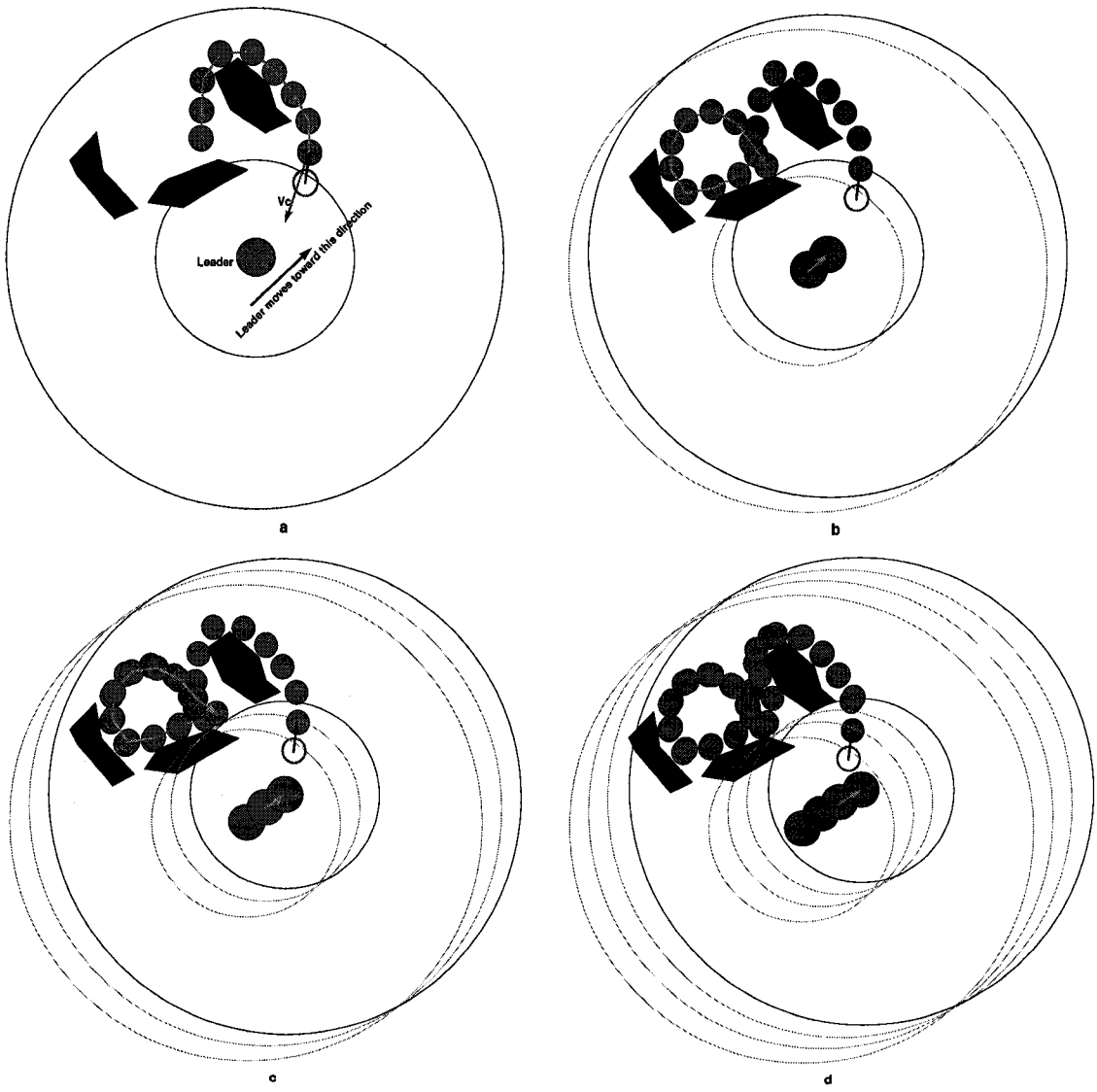


Figure 2.25: Example showing how the robot may become stuck and then freed again through the addition of a *Wandering* behavior.

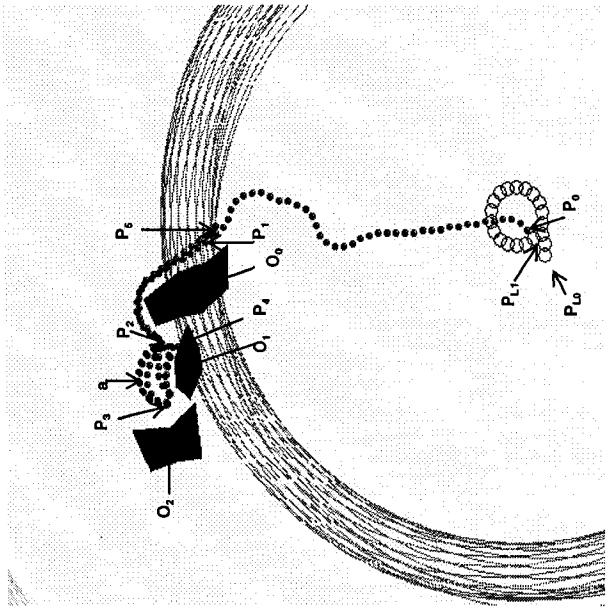


Figure 2.26: Simulation trace showing robot getting "unstuck" due to *Wandering* behavior.

respectively. We use the notation $OZA_{i \to j}$ to indicate that the robot designated to zone j is currently in zone i, for all $1 \leq i, j \leq 4$. Hence, when $i \neq j$, then the *OutOfZoneAvoidance* behavior must be enabled. For example, $OZA_{TL \to TR}$, $OZA_{BR \to TR}$ and $OZA_{BL \to TR}$ represent the *OutOfZoneAvoidance* behaviors when robot W_{TR} is in zones TL, BR or BL ,respectively. We further distinguish this behavior as being *SideOutOfZoneAvoidance* and *DiagonalOutOfZoneAvoidance* for the situations in which i and j represent adjacent quadrants or diagonal quadrants. There are exactly four situations requiring *DiagonalOutOfZoneAvoidance*; namely $OZA_{TL \to BR}$, $OZA_{TR \to BL}$, $OZA_{BR \to TL}$ and $OZA_{BL \to TR}$.

Recall that the *OutOfZone* detection mechanism of Figure 2.15 determines whether or not a robot is out of its designated zone. This network made use of the network

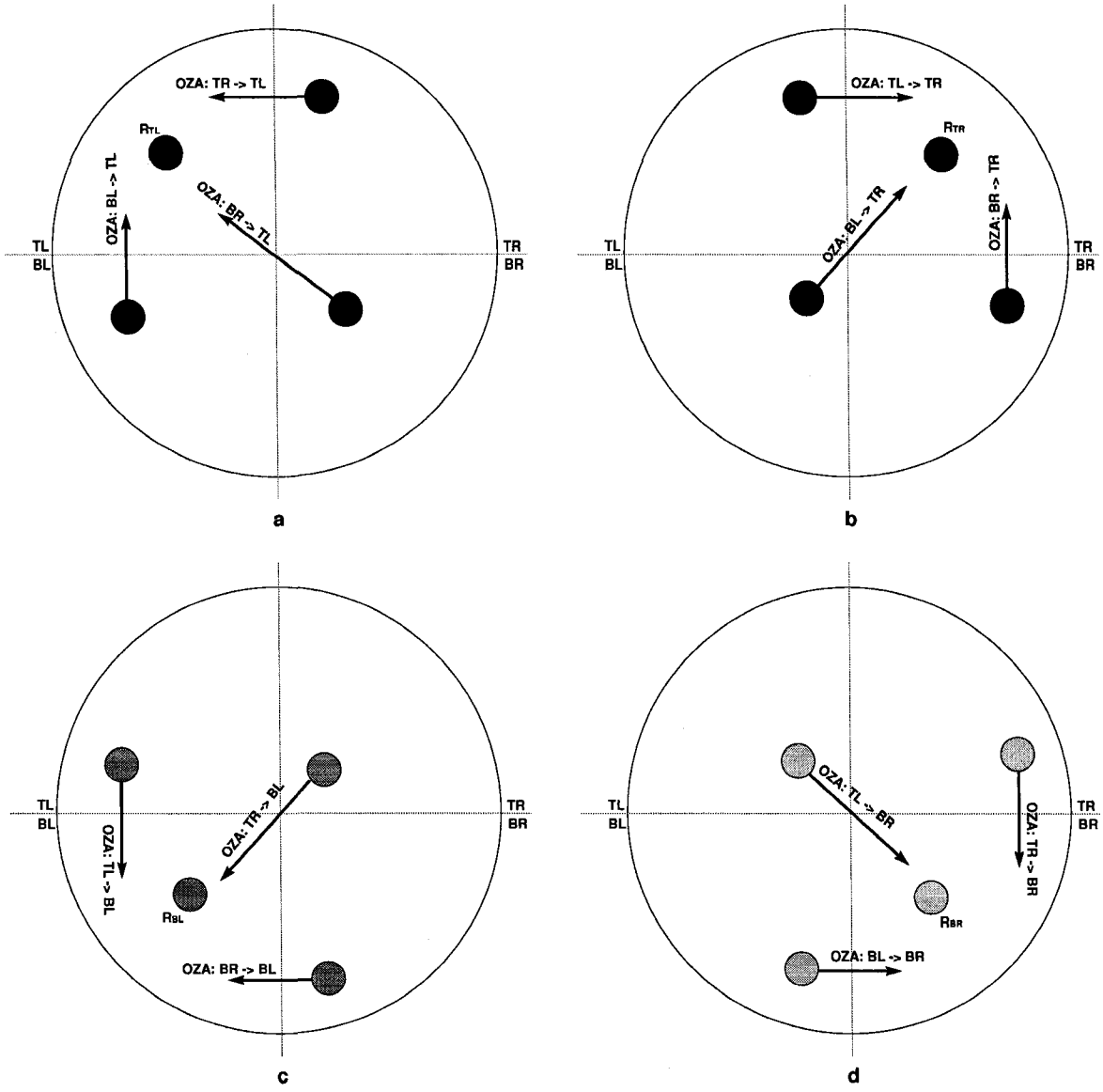


Figure 2.27: Notation used for all situations in which a robot is out of its designated Zone.

of Figure 2.14 which was based on odometry information.

For the purpose of brevity, we consider a robot that is designated to the top right zone (i.e., TR) and describe how it can find its way back to the TR zone when in one of the other three zones. Afterwards, we discussed the changes required for implementing this behavior for robots in the other zones.

To begin, we break the problem down into eight possible subproblems corresponding to the zone that the robot is currently lying in as well as the direction that it is facing. In each case, the robot will respond by turning either left or right in order to head into the appropriate direction to re-attain a position in its designated zone. Each case defines left and right direction constraint boundaries. That is, when the robot's direction lies between the two direction boundaries defined for that case, then the corresponding behavior for that case is performed. Each case is labeled **a** through **h** and is shown in Figure 2.28. The dashed line boundaries for each case indicates that this particular direction angle is not included in the case. Hence if the robot is facing the exact direction of a dashed boundary line, then this case does not apply.

Table 2.3.4 shows the turning decisions that need to be made for the robot based on its current zone and its current direction **d**. The individual turn decisions reflect the most likely direction to head in that would result in the quickest route back to the robot's designated zone.

For robots designated to the other three zones (i.e., TL, BL and BR) we can produce similar sets of range-determining networks. The ranges and turns are shown in Table 2.3.4.

Each of the ranges in Table 2.3.4 can be determined through a simple network which compares the robot's current direction to the specified range boundaries. These eight corresponding networks are shown in Figure 2.29. In this network, neuron

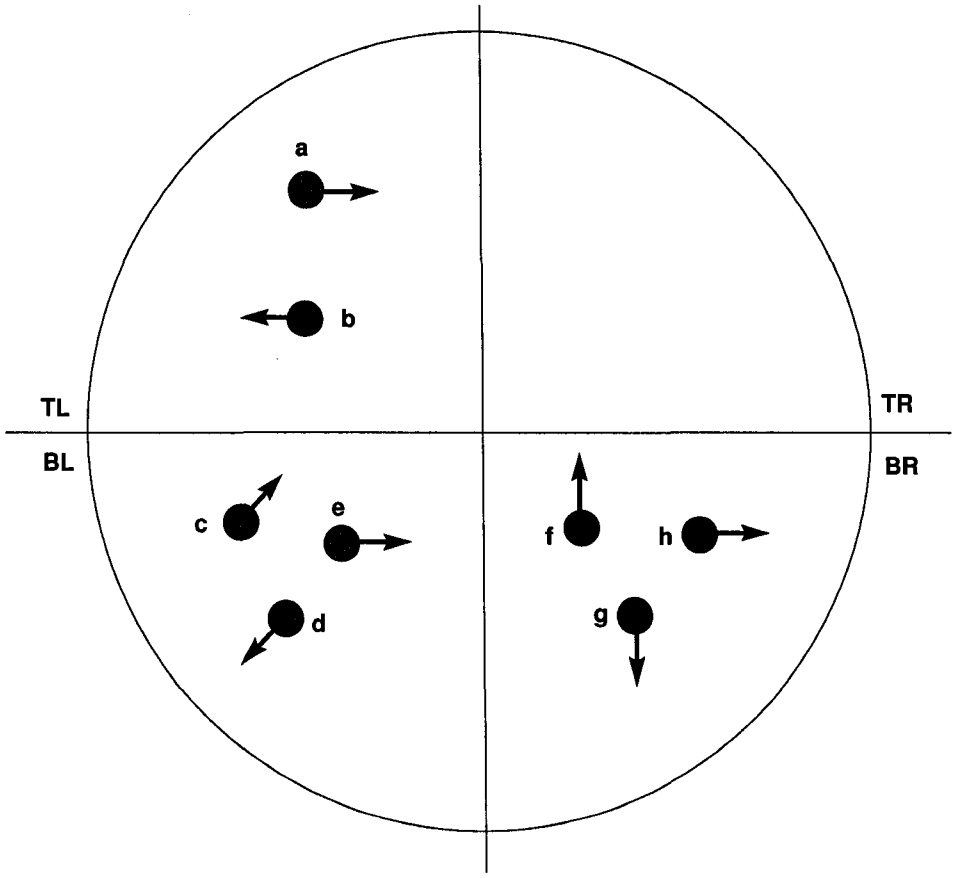


Figure 2.28: The eight *OutOfZoneAvoidance* behavior situations required used to decide which direction to turn the robot.

NewD represents the robot's current direction (i.e., from 0 to 360) and is obtained from the Odometry Nnetwork of Figure 2.10.

Notice that the bottom threshold neurons (e.g., **Range a**) are excited when the robot's direction is within the degree range. These threshold neurons are plugged into an additional network to perform the required turn and hence complete the *OutOfZoneAvoidance* behavior. This network is shown in Figure2.30.

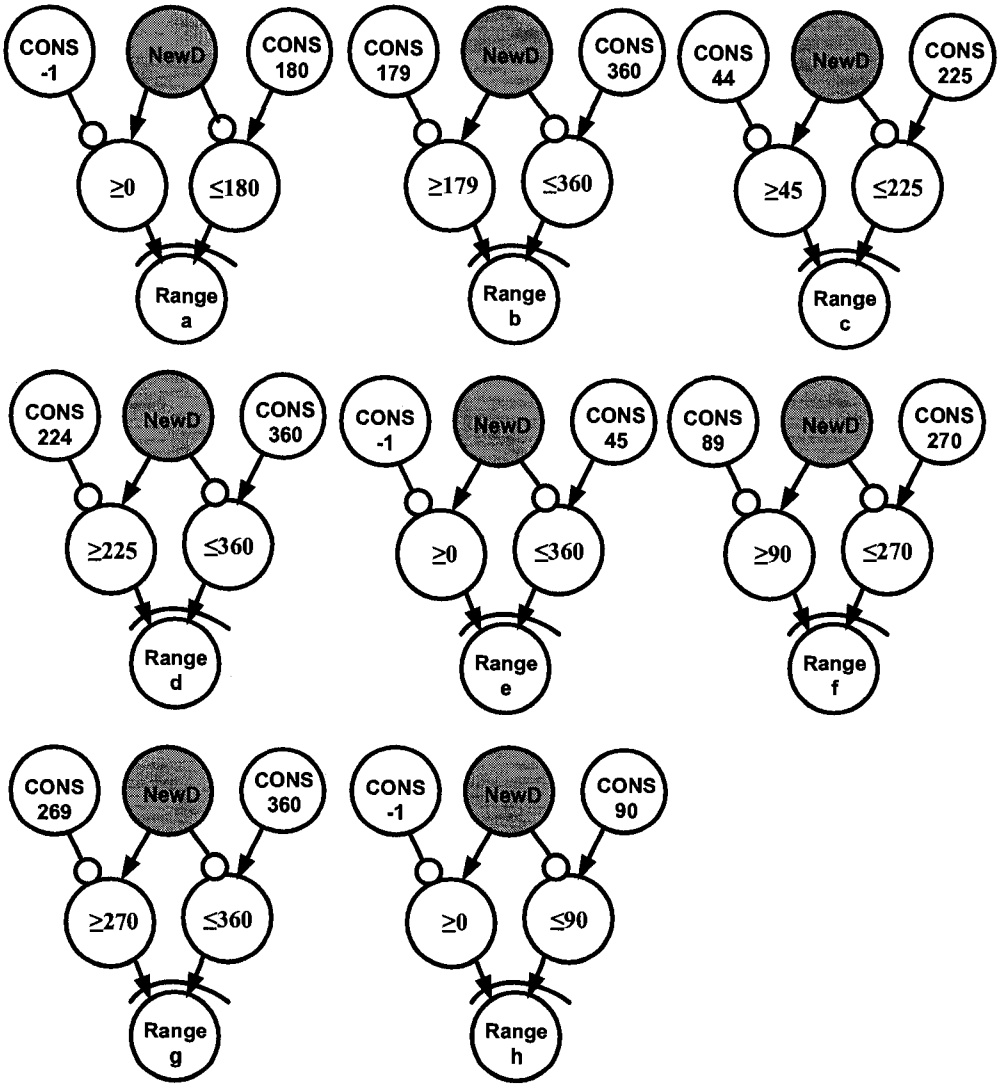


Figure 2.29: Networks for determining direction ranges for all 8 scenarios.

Avoidance Type	Range	Case	Turn
$OZA_{TL \rightarrow TR}$	$0 \leq d < 180$	a	RIGHT
	$180 \leq d < 360$	b	LEFT
$OZA_{BL \rightarrow TR}$	$45 \leq d < 225$	c	RIGHT
	$225 \leq d < 360$	d	LEFT
	$0 \leq d < 45$	e	LEFT
$OZA_{BR \rightarrow TR}$	$90 \leq d < 270$	f	RIGHT
	$270 \leq d < 360$	g	LEFT
	$0 \leq d < 90$	h	LEFT

Table 2.1: Turn decision to make for each zone and range.

The network consists of three parts which correspond to the three avoidance behaviors $OZA_{TL \rightarrow TR}$, $OZA_{BR \rightarrow TR}$ and $OZA_{BL \rightarrow TR}$ which are depicted in Figure 2.27b. Although this network corresponds solely to the robot designated to the **TR** zone, similar networks would also be constructed for robots that are designated to the **TL**, **BL** and **BR** zones.

In this network, neurons **TR**, **TL**, **BL** and **BR** come from the network in Figure 2.14. The **OutOfZone** neuron comes from the corresponding network of Figure 2.15 and is used as a means of enabling the *OutOfZoneAvoidance* behavior. The 8 **Range**[**X**] neurons come from Figure 2.29 and are used to decide whether or not the robot should turn left or right. As can easily be seen, only one of the three neurons **BR**, **TL** or **BL** are enabled at any time since the robot can only be in one zone at any time. Hence, only PartA, PartB or PartC of the network is enabled. All of these plug into the **Left** and **Right** neurons which connect to the *Steering* network of Figure 2.5 in order to direct the robot accordingly. Note that the robot continues its *OutOfZoneAvoidance* behavior until either the **OutOfZone** neuron is disabled or the **TR** neuron becomes active, indicating that the robot arrived in its designated zone. In addition, if the robot ever leaves safe communication range, the **OutOfRange**

Avoidance Type	Range	Turn
$OZA_{BL \rightarrow TL}$	$90 \leq d < 270$	RIGHT
	$270 \leq d < 360$	LEFT
	$0 \leq d < 90$	LEFT
$OZA_{TR \rightarrow TL}$	$180 \leq d < 360$	RIGHT
	$225 \leq d < 360$	LEFT
$OZA_{BR \rightarrow TL}$	$135 \leq d < 315$	RIGHT
	$315 \leq d < 360$	LEFT
	$0 \leq d < 135$	LEFT
$OZA_{TL \rightarrow BL}$	$270 \leq d < 360$	RIGHT
	$0 \leq d < 90$	RIGHT
	$90 \leq d < 270$	LEFT
$OZA_{BR \rightarrow BL}$	$180 \leq d < 360$	RIGHT
	$0 \leq d < 180$	LEFT
$OZA_{TR \rightarrow BL}$	$45 \leq d < 225$	RIGHT
	$225 \leq d < 360$	LEFT
	$0 \leq d < 45$	LEFT
$OZA_{BL \rightarrow BR}$	$0 \leq d < 180$	RIGHT
	$180 \leq d < 360$	LEFT
$OZA_{TR \rightarrow BR}$	$270 \leq d < 360$	RIGHT
	$0 \leq d < 90$	RIGHT
	$90 \leq d < 270$	LEFT
$OZA_{TL \rightarrow BR}$	$0 \leq d < 135$	RIGHT
	$315 \leq d < 360$	RIGHT
	$135 \leq d < 315$	LEFT

Table 2.2: Turn decision for each zone and range for robots designated to zones TL, BL and BR.

neuron becomes active and this also disables the *OutOfZoneAvoidance* behavior.

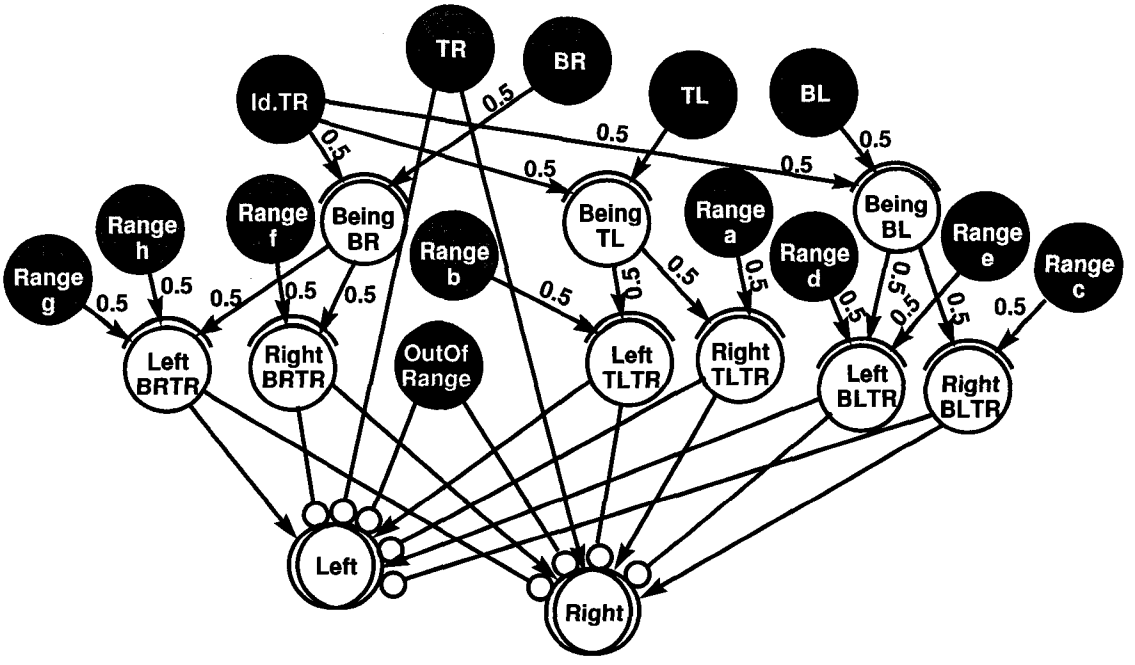


Figure 2.30: OutOfZone avoidance for robots whose zones are TR

Chapter 3

Navigation Behavior and Mechanisms

3.1 Point to Point Navigation (PTP Navigation)

A robot's overall navigation can be broken down into a sequence of *Point to Point Movements*. In our work, we accomplish this through a simple behavior that steers the robot from its current location towards some other point which is either fixed or dynamic. Each robot first determines the direction of a *relative direction vector* (RDV) towards the target point and then moves forward while continuously steering towards this direction. Figure 3.1 depicts such a *PTP Navigation* model.

In this model, the leader heads to a target point (P_{Target}) by following the relative direction β, which can be calculated based on eight transformation cases according to the direction α, that the RDV makes with the X-axis. These cases were discussed in Section 2.3.3. In fact, in this chapter, we make use of various networks defined in

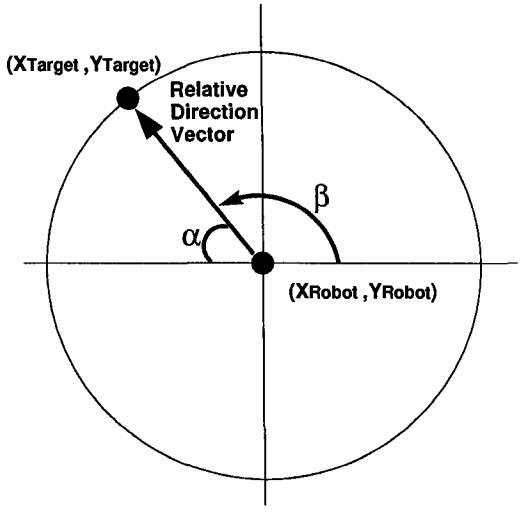


Figure 3.1: Relative direction vector model.

Chapter 2. These are as follows, **Calculate α**, **Transfer to β** and **Steer towards a direction**, these networks are those of Figure 2.18, 2.20 and 2.21.

As explained in Chapter 2, α can be determined based on Equation 3.1

$$\alpha = Atan\,(Y_{Diff}/X_{Diff}) \tag{3.1}$$

X_{Diff} and Y_{Diff}, can be calculated by applying Equation 3.2 and Equation 3.3.

$$X_{Diff} = X_{Target} - X_{Robot} \tag{3.2}$$

$$Y_{Diff} = Y_{Target} - Y_{Robot} \tag{3.3}$$

Figure 3.2 shows how α can be computed using a neuron network, which includes the template for calculating α as defined in Figure 2.18. In this network, neurons

NewX and **NewY** (coming from the odometry network), represent the robot's location. Neurons **TargetX**, **TargetY**, **XDiff** and **YDiff** represent the variables of equations 3.1, 3.2 and 3.3. The output of neuron **ATAN** denotes the angle α between relative direction and the X axis.

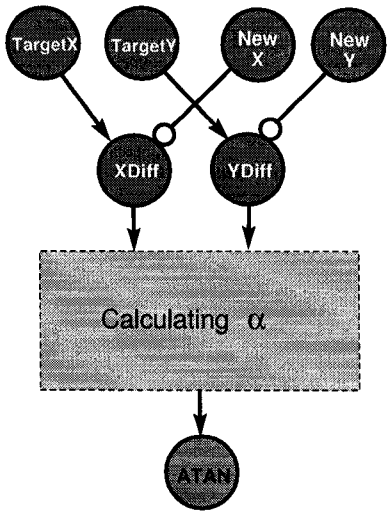


Figure 3.2: Network used to calculate relative direction α.

The eight situations for transferring α to β can be represented by the eight situations described in Figure 2.19. Based on the eight situations, we use the template of Figure 2.20 to determine the network for calculating the desired direction β. In this network (see Figure 3.3), neurons **XDiff**, and **YDiff** and *ATAN* correspond to those of Figure 3.2. Neuron **RDV** represents the value of desired direction β.

To complete the PTP navigation, robots must then steer towards the desired relative direction, β. This steering can be accomplished through the network shown in Figure 3.4 which uses the template of Figure 2.21. In this network, neuron **NewD**

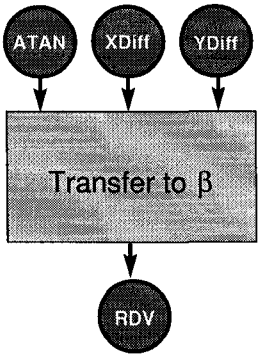


Figure 3.3: Network to compute the desired direction of DRV, β.

is from the odometry network of Figure 2.11, representing the current direction of the robot. Neuron **RDV** comes from the network of Figure 3.4, where the output represents the relative direction β. The outputs of **Left** and **Right** determine the direction a robot should steer, which plugs into the steering network of Figure 2.5.

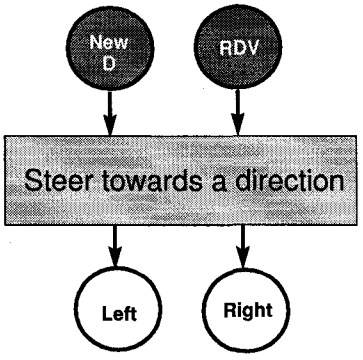


Figure 3.4: Network to turn towards the relative direction.

The two situations shown in Figure 1.2 are common to *PTP navigation.* The *Path-Concentrated Coverage* shown in Figure 1.2A contains an environment having

a fixed path along which robots can continuously execute the fixed *PTP navigation*, in which, both start point and end point are fixed. The *Centralized Concentration Coverage* model consists of invisible target points, that are not predetermined. Robots can only determine where to move according to the dynamic integrated information provided by either the outer environment or robots themselves. In contrast to *Fixed PTP Navigation* (i.e., Path-Concentrated), our solution to implement *Dynamic PTP Navigation* (i.e., Centralized Concentration) is more complicated. In such dynamical PTP navigation, some dynamic computing rules are set up as vectors, which help robots to integrate and extract the useful information. In the following sections, we explained both *Fixed PTP Navigation* and *Dynamic PTP Navigation* models in detail.

3.2 Fixed PTP Navigation ($FPTP$)

Consider the example in Figure 3.5, in which a leader robot has exactly 4 worker robots and must travel to some goal (i.e., target) points sequentially. Without loss of generality, assume that there are 4 goal locations, one in each of the 4 areas as shown. These goal locations represent some fixed path that we would like the robots to travel on. The movement of the leader robot along this fixed path allows us to steer the entire robot hierarchy along the path as well. In order to accomplish this type of navigation, we make use of the PTP navigation mechanism (desired in the previous section) to steer towards one of the goals. We then use a "Goal Switching" mechanism to decide upon the next goal in the sequence.

In this thesis, the "Goal Switching" mechanism is enabled each time the leader has verified that it reached a goal. To check whether the robot has arrived at a

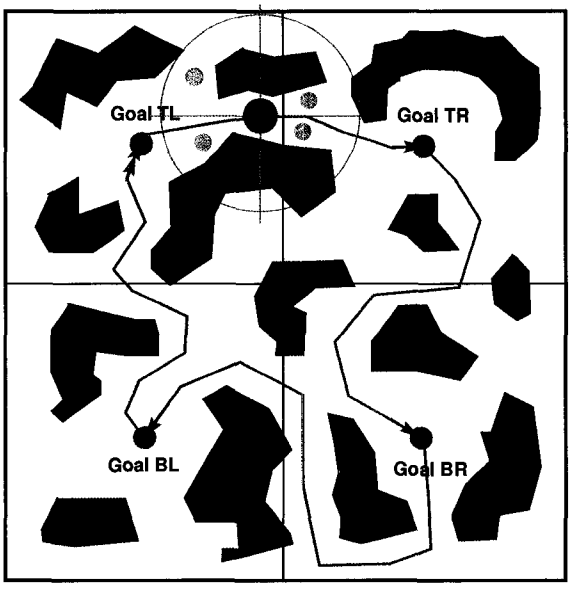


Figure 3.5: FPTP navigation model.

goal, we compare the robot's location with the goal location to determine if the difference in X and Y coordinate values is within some threshold (which was chosen as 12 measurement units (i.e., the leader robot's radius) in our experiments). This threshold is hard coded into the network, along with the goal locations. This value can be user defined parameter which may vary according to the robot's size, sensor position and odometry accuracy.

Figure 3.6 shows the "Goal Judging" network to judge whether a robot has arrived at $Goal_{TR}$. As before, neurons **NewX** and **NewY** come from the odometry network and represent the robot's current position. **GoalXTR** and **GoalYTR** are constant neurons representing the coordinates of the current goal position.

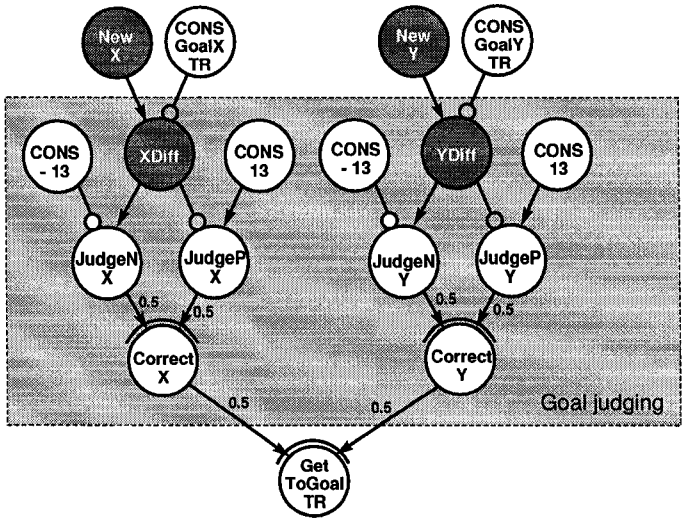


Figure 3.6: "Goal Judging" network for the TR zone.

As can be seen in the network, the **CorrectX** and **CorrectY** neurons determine whether the robot is within the fixed threshold distance of the goal (i.e., 12). The **GetToGoalTR** neuron is then excited to indicate that the robot has reached the goal. The other three goals $Goal_{TL}$, $Goal_{BL}$ and $Goal_{BR}$ use the same "Goal Judging" template, as shown in Figure 3.7. Hence one such network is needed per goal location. The example represents a "proof of concept" for our network. We do not explain how these fixed goal locations are chosen. This may be user-defined parameters. If the robot is to operate in a fixed/static environment, these constant goal neurons may be configured once. In a dynamic environment, it may be necessary to change them over time. Perhaps some fixed number of such goal networks can be implemented to allow some maximum number of goal points to be specified, leaving unused ones to be disabled as necessary. We leave this as future work.

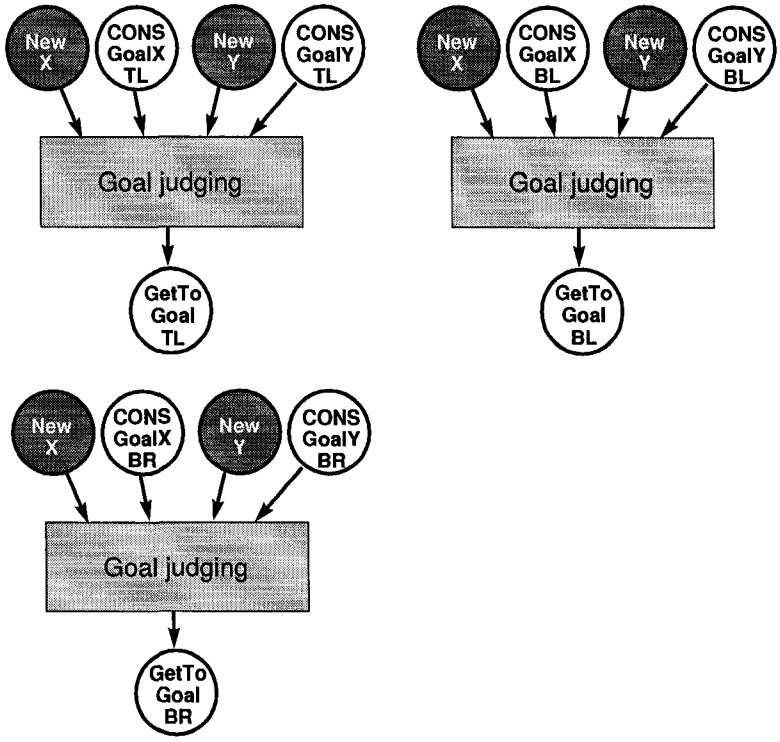


Figure 3.7: "Goal Judging" networks for zones TL, BL and BR.

Once these individual "Goal Judging" networks have been implemented (one per goal), they need to be combined into a network that switches between them. Such a network is shown in Figure 3.8. Finally, the **TargetX** and **TargetY** neurons feed into the network of Figure 3.2 to steer the robot towards the desired point along the fixed path.

The four **GetToGoal** threshold neurons along the top right side of the figure come from the individual "Goal Judging" networks just described. The four **Index** sustain neurons are combined with the threshold neurons so that they are activated in sequence as the robot reaches each of the goals in turn. The **Init** pulse neurons

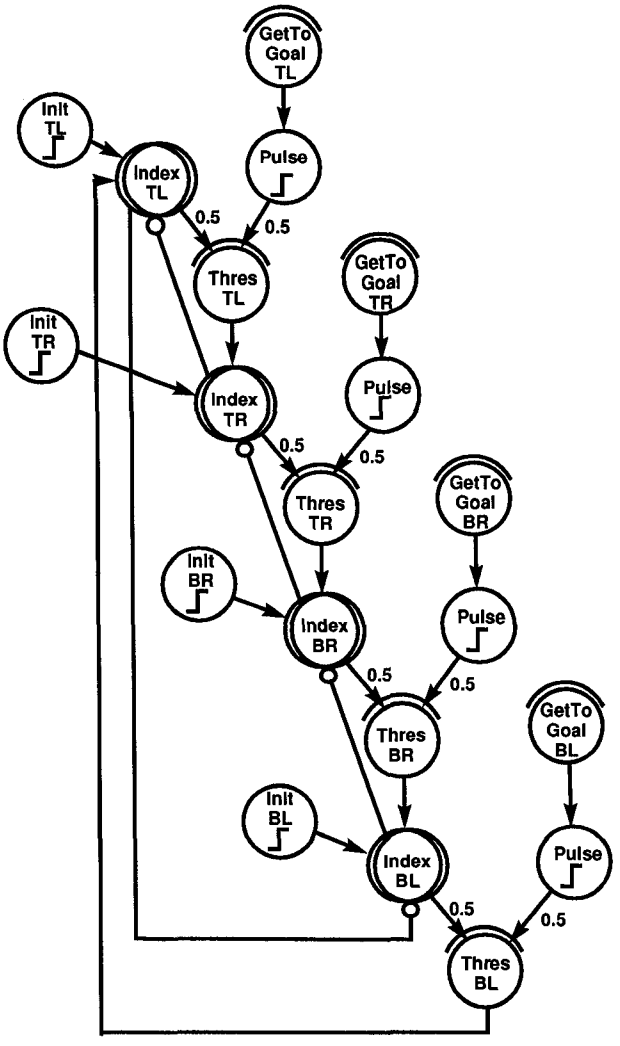


Figure 3.8: Overall "Goal Judging" network for a 4 point path.

along the left side of the network allow the user to enable the starting goal location. Only one of these should be used. If the user always knows the initial goal location, then only one of these pulse neurons is necessary.

The network of Figure 3.9 is then used to steer the robot to the appropriate goal location. Here, the constant goal location neurons correspond to those from Figures 3.6 and 3.7. The **Index** sustain neurons corresponding to those of Figure 3.2 produce the relative direction vector that heads to the target.

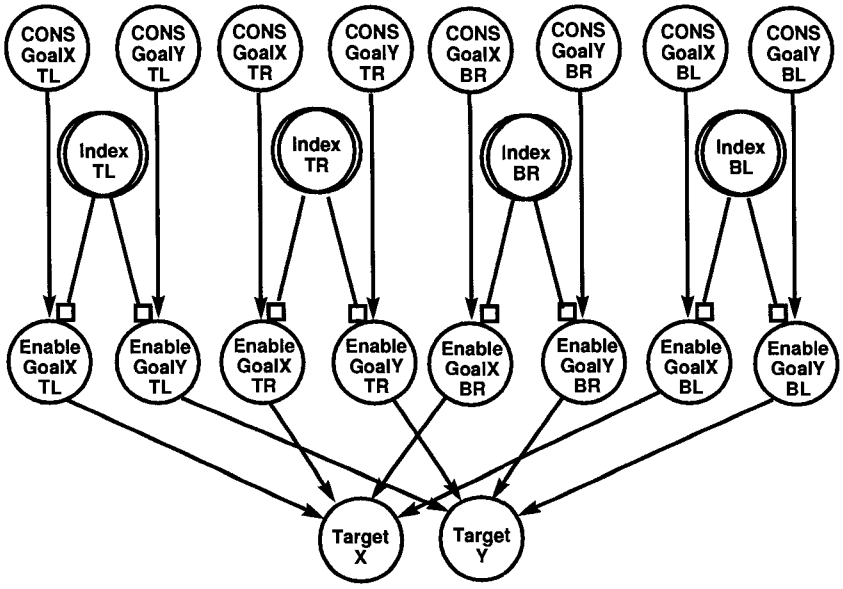


Figure 3.9: Network for choosing the appropriate goal location.

3.3 Dynamic PTP Navigation

FPTP Navigation requires setting up a model of the environment and manually planning/hard coding a path. For exploration missions in a dynamic or even unknown

environment, robots can only dynamically navigate or explore relying on the previous knowledge about the environment or their own location history. In the following section, such a dynamic navigation model, namely, *Previous Knowledge based Navigation (PKN)* is explained in detail.

The PKN approach introduced in this thesis is a completely self-led navigation model within which, robots make navigation decisions to explore unknown areas and also avoid obstacles, based solely on its previous locations. In this thesis, we explained three types of PKN approaches by defining 3 vectors: *Previous Position Vector*, *OutOfRangeCount Vector* and a *Combined Vector*. In each case, the vectors direct the robot towards a moving goal location. The term *Vector* is used because the model then steers the robot from its current location towards the dynamically changing goal point, resulting in a relative direction vector of unit magnitude. In the following sub sections, each of these are explained.

3.3.1 Previous Position Vector (PPV)

The PPV is a vector that represents the average of the robots' recently traveled locations. In this thesis, *Global Average Position (GAP)* and *Last Hundred Average Position (LHAP)* models are presented. In GAP, the global average position is calculated based on all of the previous traveled locations as Equations 3.4 and 3.5 show. In the equations, *Times* represents the number of time units (i.e., global counter) since the robot was dispatched from its leader. In $LHAP$, the local average position is calculated based on the most recent hundred traveled locations. Equation 3.6 and Equation 3.7 show the process of calculating X_{LHAP} and Y_{LHAP}.

$$X_{GAP} = \sum_{i=0}^{Times-1} X_i/Times \tag{3.4}$$

$$Y_{GAP} = \sum_{i=0}^{Times-1} Y_i/Times \tag{3.5}$$

$$X_{LHAP} = \sum_{t}^{t+99} X_i/100 \tag{3.6}$$

$$Y_{LHAP} = \sum_{t}^{t+99} Y_i/100 \tag{3.7}$$

Since GAP represents the average position of all previously covered points, we can use the network of Figure 3.10 to calculate the GAP.

As before, the **NewX** and **NewY** are from the odometry network, and represent the robot's current location. Two accumulative neurons **AccumX** and **AccumY** represent the sum of all previous locations. **TimeCount** counts the total time the robot has been active. The **DIV** module calculates $1/Times$. **XGAP** and **YGAP** are two product modules used for calculating X_{GAP} and Y_{GAP} as explained in Equations 3.4 and 3.5.

The $LHAP$ scheme was proposed in order to investigate any advantages that it might have over the global average GAP scheme. Therefore, we did not develop a network for maintaining the previous 100 robot locations from which we would obtain the average. Instead, it was implemented as an array of values which were fed into a summing routine which was then the used to connect to a simple network to find

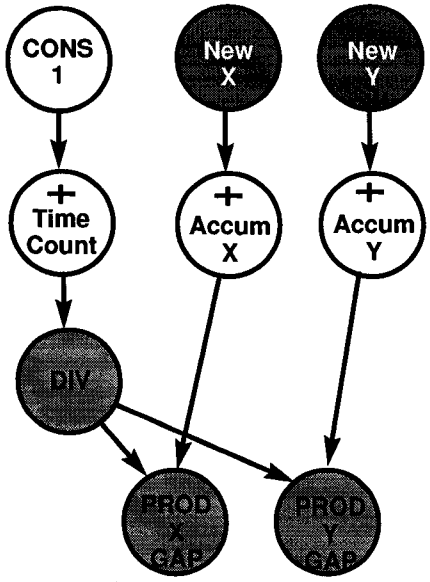


Figure 3.10: Network to calculate the *GAP*.

the average of the 100 points. However, it is possible to construct such a network to store 100 previously encountered locations and average them. The details have been left out of this thesis, for the sake of brevity.

The actual vectors for *GAP* and *LHAP* are computed so as to point away from the computed global or local average (see Figure 3.11). This allows the robot to steer away from locations previously covered (see Figure 3.12) which should result in less over-coverage and aid in regards to implementing an outwards search strategy.

It is also possible to head "towards" the global average, which allows the robot to stay near to where it has been. The vector in this case would be negated and is denoted as *OGAPV*. In regards to the *LHAP* strategy, the local average position vector (LHPV) would either steer the robot towards or away from locations that it has

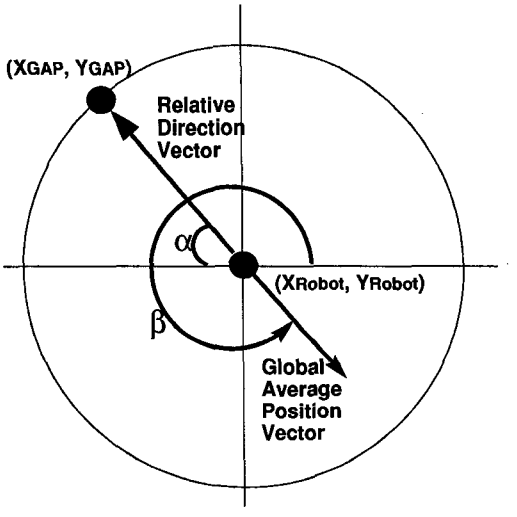


Figure 3.11: Model of the PPV.

visited recently. This differs from the global average strategy since the GAP vector tends to point to the "center" of a robot's previous locations whereas the $LHAPV$ does not centralize over time (see Figure 3.13).

The manner in which robots determine the direction of the $GAPV$ and $LHPV$ is the same as that used in determining the **RDV** discussed previously (see Figure 3.3). Figure 3.14a, b and c show the networks implementing the mechanism to calculate untransferred direction of $GAPV$, $LHAPV$ and $OGAPV$, which use the template of Figure 3.2 to calculate α. In this network, **XGAP** and **YGAP** represent the global average position from the network of Figure 3.10.

We apply the $GAPV$, $LHAPV$ and $OGAPV$ to our PTP navigation pattern. Different coverage results can be obtained through changing the weights W_{GAP}, W_{LHAP} and W_{OGAPV} of Figure 3.14. For example, we can change W_{GAP} to control how far

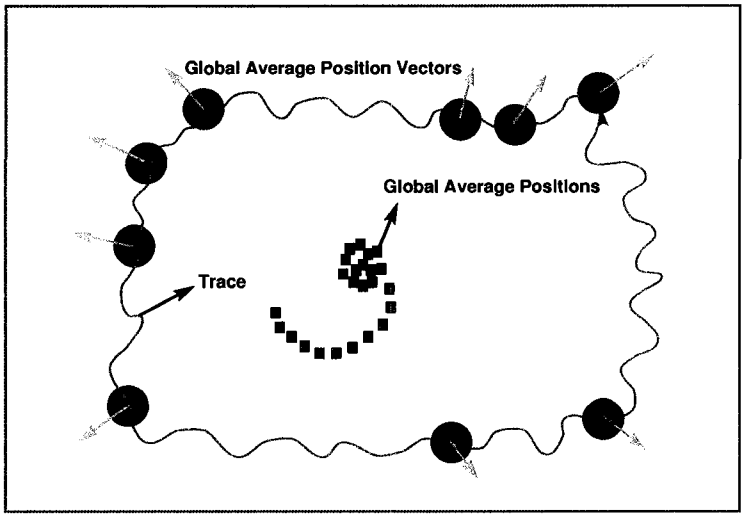


Figure 3.12: Outwards expansion property of the *GAPV*

the worker robots should expand their searching area when preforming a *Centralized Concentration Coverage* pattern.

3.3.2 Out Of Range Count Vector (*ORCV*)

In contrast to the *GAPV*, the *ORCV* model described here helps robots to make decisions based on previous information generated by the robots themselves. It is easier for worker robots to leave the safe communication range when operating in a complex environment than when in a simpler environment. Thus, counting the times that a worker robot travels out of range is a good way to determine or predict obstacle-cluttered areas. *ORCV* can be set up through four sub-vectors, each counting the number of times a worker robot has been out of safe communication range in their own zone respectively.

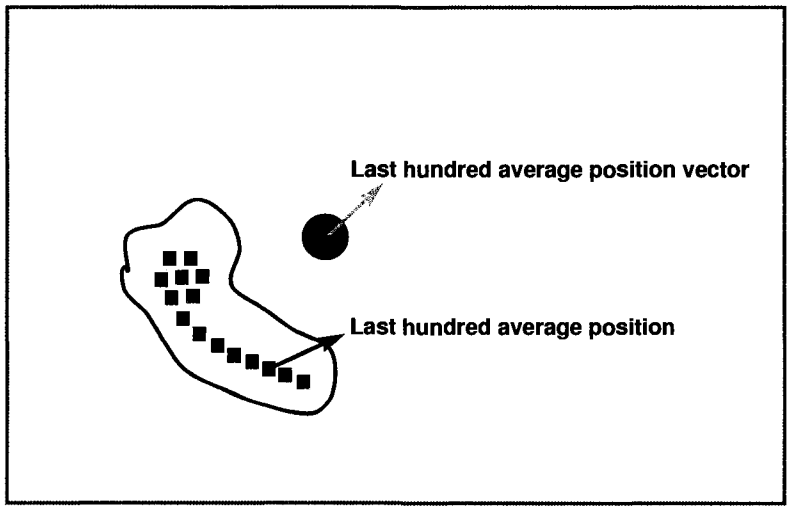


Figure 3.13: Non-centralized property of the $LHAPV$.

Figure 3.15 shows such a model. As shown in this model, the attribute Mag of each of the four vectors V_{TR}, V_{TL}, V_{BL} and V_{BR} represents the number of times each worker robot travels out of range in its corresponding zone. $V_{OutOfRange}$ is integrated from two vectors V_{BRTL} and V_{TRBL}, that are combined from V_{BR}, V_{TL} and V_{TR}, V_{BL} respectively. Initially, $V_{OutOfRange}$ represents the direction that is most likely to have less obstacles since there were more "OutOfRange" occurrences in that direction. It is assumed that a robot who has been out of range more often has not been delayed or bounded by obstacles it may have encountered in its zone. That is, it should represent the direction to head in order to move the robot into a "more open" area.

The direction, α can be computed using $X_{OutOfRange}$ and $Y_{OutOfRange}$ as shown in Equation 3.8.

$$D_{OutOfRange} = arcTan(Y_{OutOfRange}/X_{OutOfRange}) \tag{3.8}$$

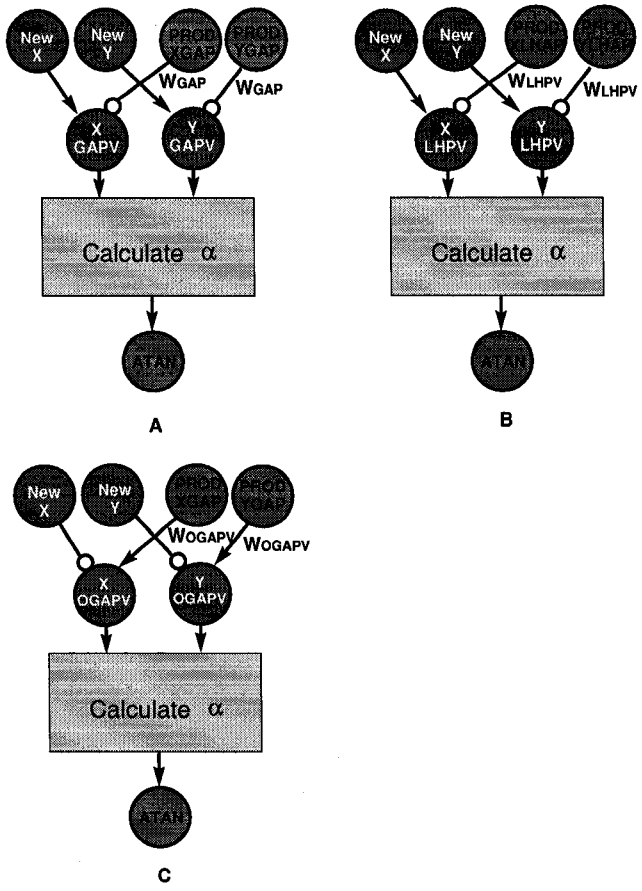


Figure 3.14: Calculating the untransferred direction α of (A) *GAPV*, (B) *LHAPV* and (C) *OGAPV*

Also, $X_{OutOfRange}$ and $Y_{OutOfRange}$ can be calculated based on Equation 3.9 and Equation 3.10.

$$X_{OutOfRange} = X_{BRTL} + X_{TRBL} \tag{3.9}$$

$$Y_{OutOfRange} = Y_{BRTL} + Y_{TRBL} \tag{3.10}$$

Here, X_{BRTL} and Y_{BRTL} are the combined X and Y coordinates from V_{BR} and

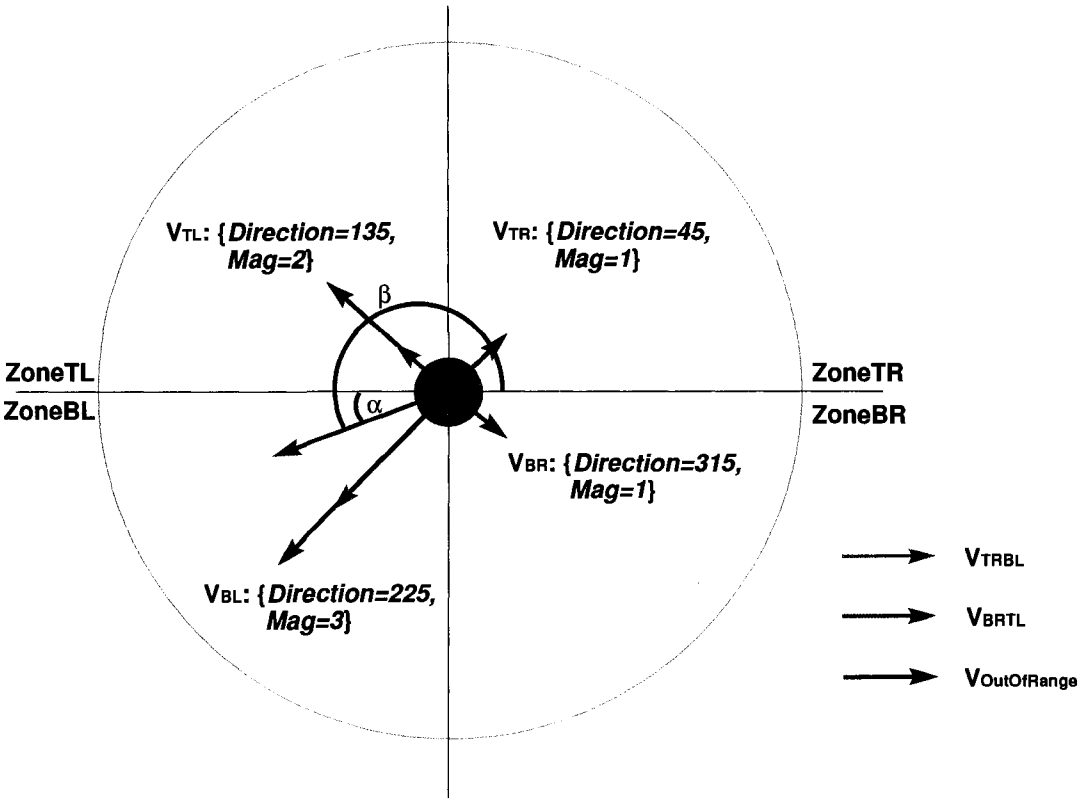


Figure 3.15: *ORCV* calculation model.

V_{TL} and X_{TRBL} and Y_{TRBL} are the combined X and Y coordinates from V_{TR} and V_{BL}. X_{BRTL}, Y_{BRTL}, X_{TRBL} and Y_{TRBL}, can all be calculated based on Equations 3.11, 3.12, 3.13 and 3.14.

$$X_{BRTL} = Mag_{BRTL} \times COS\,(D_{BRTL}) \tag{3.11}$$

$$Y_{BRTL} = Mag_{BRTL} \times SIN\,(D_{BRTL}) \tag{3.12}$$

$$X_{TRBL} = Mag_{TRBL} \times COS\,(D_{TRBL}) \tag{3.13}$$

$$Y_{TRBL} = Mag_{TRBL} \times SIN\,(D_{TRBL}) \tag{3.14}$$

Where $D_{BRTL} = 315°$, and $D_{TRBL} = 45°$. Mag_{BRTL}, Mag_{TRBL} denote magnitudes of the two diagonal combined vectors (V_{BRTL} and V_{TRBL}). They can be calculated based on the Equation 3.15 and Equation 3.16.

$$Mag_{BRTL} = |Mag_{BR} - Mag_{TL}| \tag{3.15}$$

$$Mag_{TRBL} = |Mag_{TR} - Mag_{BL}| \tag{3.16}$$

Mag_{BR}, Mag_{TL}, Mag_{TR} and Mag_{BL} are magnitudes of four sub vectors and are equal to the number of times a worker robot has been out of safe communication range in each zone. Figure 3.16 gives the network for computing the number of times the worker robots have been out of range. In this network, neurons **TL**, **TR**, **BR** and **BL** are from the network shown in Figure 2.3. These four binary neurons, when excited, indicate that a worker robot is in this zone currently.

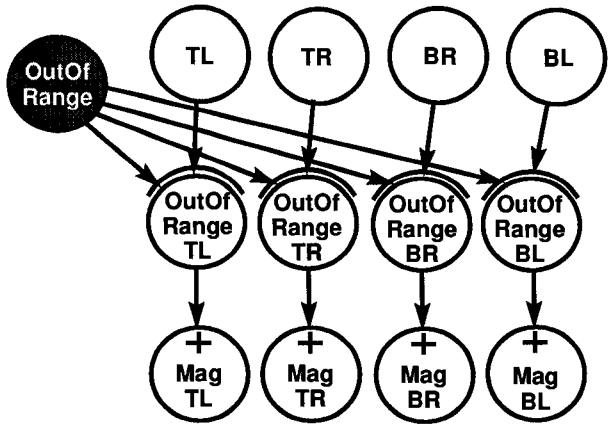


Figure 3.16: Network to calculate the out of range times for worker robots.

Figure 3.17 shows the network used to calculate the untransferred direction α of $V_{OutOfRange}$. In this network, neurons Mag_{TL}, Mag_{TR}, Mag_{BR} and Mag_{BL} are from

Figure 3.16 representing the magnitude of the four sub vectors. The remaining part of this network is divided into two main portions. Specifically, **Part A** calculates the value of Mag_{BRTL} and Mag_{TRBL} and **Part B** determines the values of X_{BRTL}, Y_{BRTL}, X_{TRBL} and Y_{TRBL}.

In **Part B**, neurons **SubTRBL** and **SubBRTL** represent the value of $Mag_{TR} - Mag_{BL}$ and $Mag_{BR} - Mag_{TL}$, respectively. Neurons **ThresTRBLPos**, **ThresTRBLNeg**, **ThresBRTLPos** and **ThresBRTLNeg** are used to judge the sign of the difference between two magnitudes.

Direct45, **Direct225**, **Direct315** and **Direct135** determine the direction of $V_{OutOfRange}$. **DirectTRBL** represents D_{TRBL} in Equation 3.13. **DirectBRTL** represents D_{BRTL} in Equation 3.11 and 3.12. **XTRBL** and **YTRBL** output X_{TRBL} in Equation 3.13 and Y_{TRBL} in Equation 3.14. The **XBRTL** and **YBRTL** modules output X_{BRTL} in Equation 3.11 and Y_{BRTL} in Equation 3.12. **XOutOfRange** and **YOutOfRange** output $X_{OutOfRange}$ and $Y_{OutOfRange}$ in Equation 3.9

The neurons **XOutOfRange** and **YOutOfRange** then plug into the network of Figure 3.2 in order to calculate the relative direction vector which steers the robot towards the most likely opening.

3.3.3 Combined Vector (CV)

The CV is a combination of both $ORCV$ and $GAPV$ or $LHAPV$. The resulting vector represents a compromise between the two schemes which will attempt to steer the robot into open areas while also avoiding (or attracting to) the area that it has already covered. Figure 3.18 shows the model of this CV.

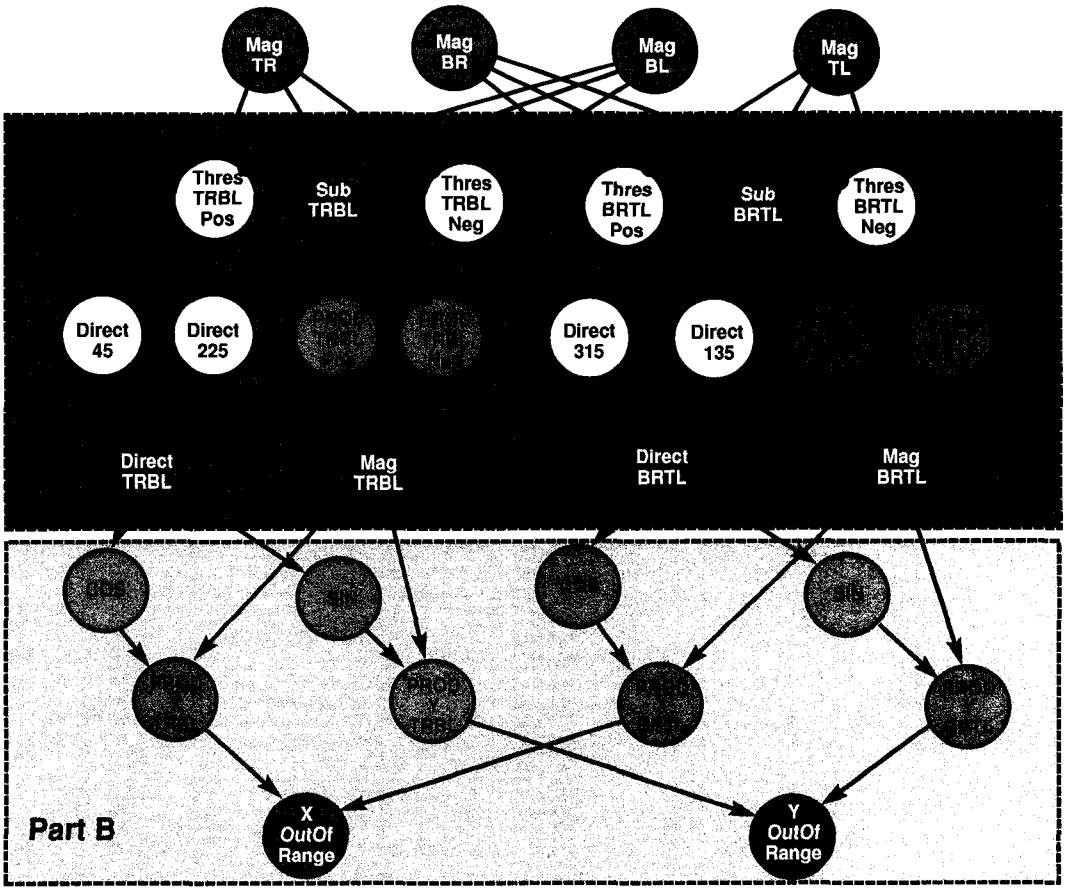


Figure 3.17: Network for calculating the out of range vector.

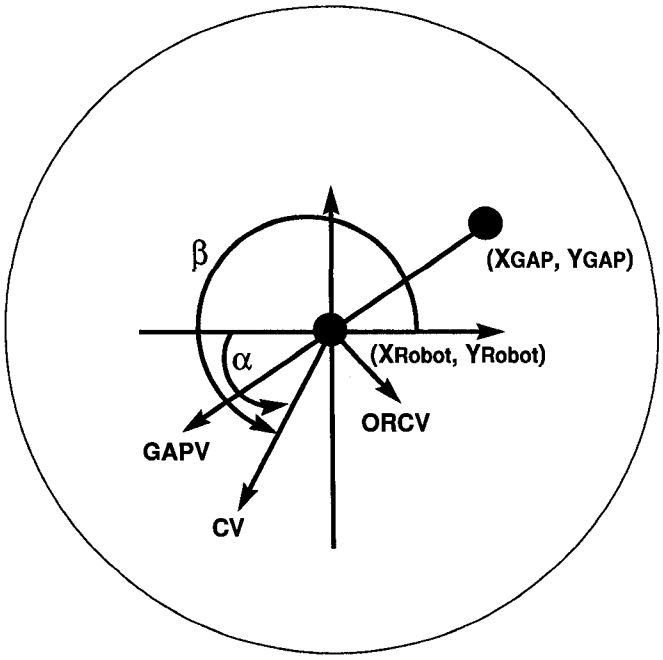


Figure 3.18: The CV calculation model.

Assume that $GAPV$ and $ORCV$ have already been determined. As shown in this model, we must first calculate the untransferred relative direction α (Equation 3.17) and transfer it into the actual direction β.

$$\alpha = arcTan(Y_{CV}/X_{CV}) \tag{3.17}$$

Y_{CV} and X_{CV} are the difference in CV and robot's current position. These two values can be calculated based on Equation 3.18 and Equation 3.19.

$$X_{CV}\,(X_{Diff}) = X_{GAPV} + X_{OutOfRange} \tag{3.18}$$

$$Y_{CV}\,(Y_{Diff}) = Y_{GAPV} + Y_{OutOfRange} \tag{3.19}$$

$X_{OutOfRange}$ and $Y_{OutOfRange}$ are values from Equation 3.9 and 3.10.

Figure 3.19a shows how we combine the networks to steer away from the global average while steering towards the most open areas. The weights on these incoming edges allows adjustments for the strength of each vector. The output of the network feeds into the relative direction vector network of Figure 3.3.

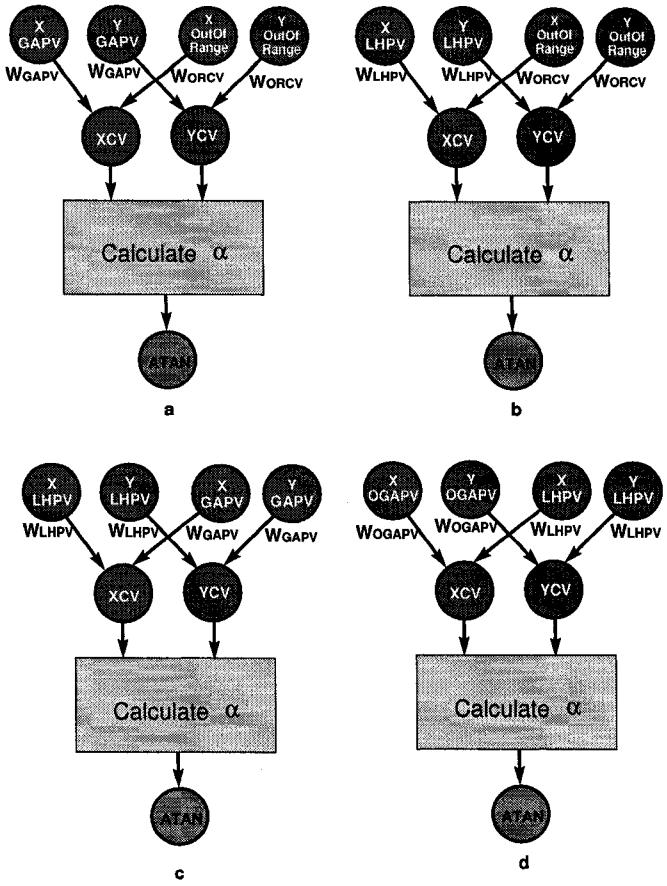


Figure 3.19: Networks to compute the untransferred direction α of CV.

The combined vector can be used to direct a robot to accomplish an *Outwards-expanding search pattern* (see Figure 1.2B). Through changing the weights, W_{GAPV} and W_{GAPV}, the influence of the $GAPV$ can be changed. In one of our experiments,

we increased W_{GAPV} four times in sequence from -0.5 to 1 to 1.5 and finally to 2.0. Test results are shown in Chapter 4.

In addition, there are other ways to combine vectors, such as that composed by $ORCV$ and $LHAPV$ or by $LHAPV$ and $GAPV$, or $ORCV$ and $OGAPV$. Figure 3.19b c and d show the networks for calculating α for these other combined vectors.

Chapter 4

Experimental Results

In this chapter, we consider experimental results for both our fixed and dynamic *Point-to-Point* navigation strategies. Through displaying and analyzing the test results, the advantage of HCSMRS in specific scenarios can be further shown. In Section 4.1, we introduce our test methodology, including the HCSMRS robot compositions, our test environments, test case descriptions, and visualization strategy. In Section 4.2, we explain and compare the test results after applying two combined vectors, $GAPORCV$ and $LHAPORCV$. In Section 4.3 and 4.4, two specific test scenarios, *Centralized Concentration Coverage* and *Path-Concentrated Coverage* were further discussed.

4.1 Test Methodology

The purpose of the experiments is to show the advantage of HCSMRS over other organized multi-robot systems (MRS). In order to demonstrate the advantages

of HCSMRS, for each test scenario, we also tested the distributed MRS (i.e., non-hierarchical MRS). In each test, there is a specific goal that hierarchy must achieve, namely to cover the environment with worker robots who perform some unspecified task.

For general coverage, the non-hierarchical MRS provides a better coverage than HCSMRS, since the robots move randomly, tending to cover the environment uniformly. However, for scenarios that require concentrated coverage in specific areas, such as those shown in Figure 1.2, HCSMRS should provide an advantage over the non-hierarchical MRS.

In this thesis, we only tested a single layer hierarchical architecture. A multi-layer hierarchy can be set up recursively based on the single layer structure, since the working mechanisms for multi-layer MRS are the same as the single layer MRS. Therefore, we believe that the single layer MRS will exhibit the same properties and advantages as a multi-layer HCSMRS.

We paid more attention on the two test scenarios shown in Figure 1.2, namely, *Path-Concentrated Coverage* and *Centralized Concentration Coverage*. In these two scenarios, robots need to focus on specific areas/locations within a limited time to obtain a "better" coverage. A non-hierarchical scheme cannot focus the coverage on any specific areas, and hence may not provide adequate coverage.

In order to further investigate the effectiveness of the HCSMRS's concentrated coverage, we aimed to compare it against the non-hierarchy MRS in various environments both with and without obstacles. We wanted to determine the effects that obstacles

have our scheme's ability to provide concentrated coverage. Figure 4.1 shows the various arrangements of obstacles used in our tests.

4.1.1 Test Setup

Our tests were run within a 1024×1024 pixel square environment. This proved to be adequate in size since we tested only a single layer hierarchy. Our environments contain 0 or more polygonal obstacles and target points. The single layer HCSMRS communication group consists of a leader robot and four worker robots. Each of the four worker robots has been assigned to a different zone (i.e. quadrant). The non-hierarchical MRS is composed of four distributed robots, that move randomly, without being constrained by any communication range or led by a leader. We consider the speed of worker robots to be the lowest time unit. That is, at each global time step, the worker robots advance one unit, whereas the leader robot advances only once every 10 time steps. The running time for all our tests was set to 20,000 time steps. For all the tests, the leader robot starts from the center point.

For our *Path-Concentrated Coverage* tests, the leader continuously follows the predetermined path, $Goal_0 \rightarrow Goal_1 \rightarrow \cdots \rightarrow Goal_7$. Figure 4.1B shows the locations of obstacles and goals for this test case. For the *Central Concentration Coverage*, we tested our HCSMRS within an empty environment as well as with the obstacle-cluttered environment shown in Figure 1.1.

Since our mechanisms for performing both *Path-Concentrated Coverage* and *Centralized Concentration Coverage* make use of various vector-based navigation strategies, we performed some additional experiments which were aimed at comparing the various strategies and the parameters involved.

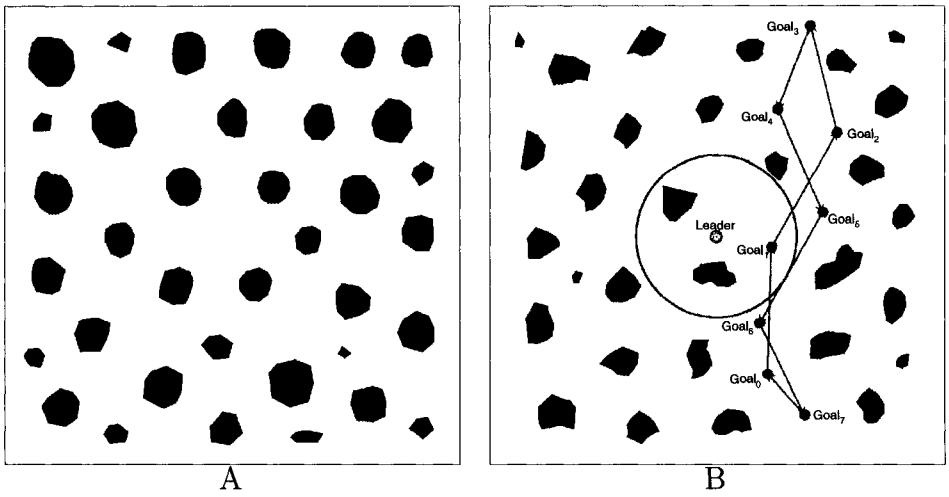


Figure 4.1: Obstacle arrangements for our tests.

We display the coverage results from our tests for both the worker and leader robots. As shown in Figure 4.2A, a *Location Coverage* is an image showing each location that was covered by the worker robots. In addition, to show the leader robot's coverage, we display the trace of the leader robot as red circles, each representing a position that the leader robot has covered (see Figure 4.2B).

We produced a *Generalized Coverage Map*, which is a classification of the *Location Coverage* through use of a gray scale image. The map's purpose is to highlight the neighborhoods that have been covered more often, giving lighter color to the parts having less coverage. Moreover, since small details shown in *Location Coverage* are unimportant to us, the *Generalized Coverage* map makes it easier to analyze the overall coverage. Figure 4.3 shows the comparison between both the *Location Coverage* and its corresponding *Generalized Coverage Map.*

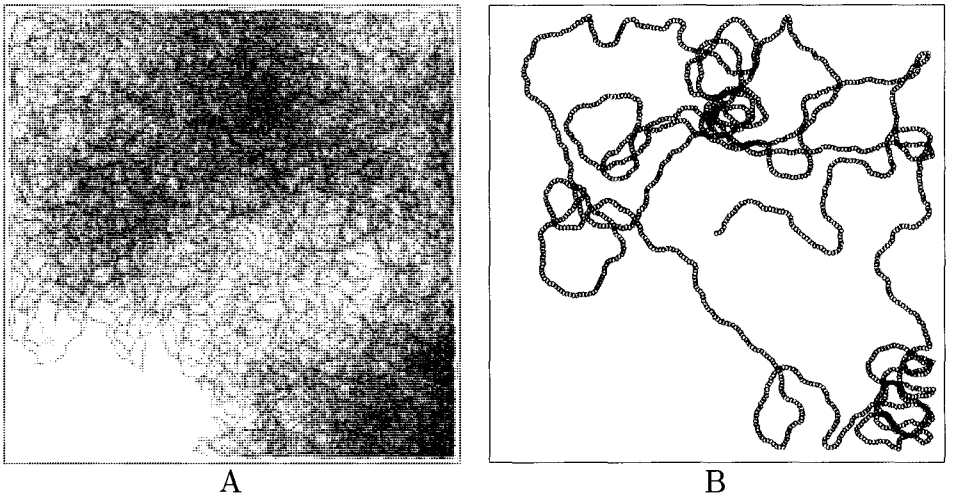


Figure 4.2: The results from random movement tests. (A) The *Location Coverage*. (B) The leader's trace.

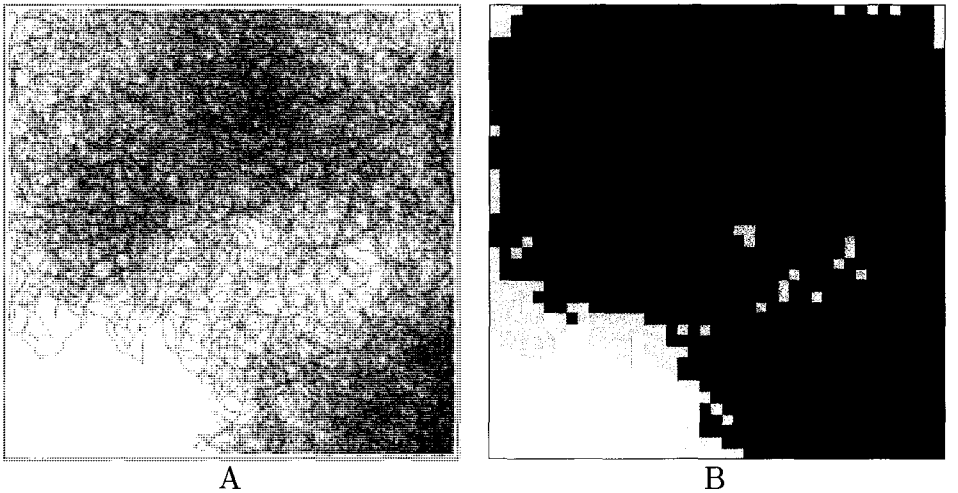


Figure 4.3: Comparison of (A) *Location Coverage* and (B) *Generalized Coverage Map* made from Random Movement tests.

To generate this *Generalized Coverage Map*, we first count the times that each pixel has been covered during the test, as shown in the left of Figure 4.4A. We save these values into a 1024×1024 array, where each array element corresponds to the number of times that a location in the environment was covered by a worker robot. Then, we define an $S_{Merge} \times S_{Merge}$ neighborhood, where S_{Merge} is some adjustable constant. In Figure 4.4, for example, $S_{Merge} = 2$, but in our test cases, we use $S_{Merge} = 16$. The following algorithm illustrates the process of generalizing the pixel coverage to produce a merged array, which represents the *Generalized Coverage Map*. It should be noted that various smoothing filters and other image processing techniques can be applied to this data beforehand. However, we are interested only in getting a rough idea as to the coverage of the worker robots.

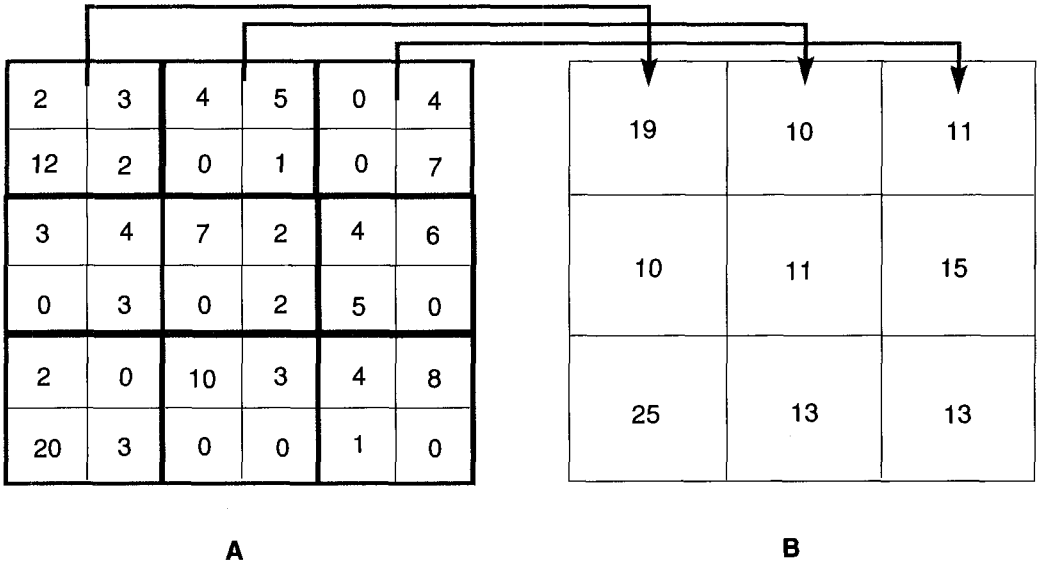


Figure 4.4: Example of pixels with its coverage times.

Algorithm: Merge the covered counts of pixels using a 16×16 neighborhood.

Input : A 1024×1024 two dimensional array, **Pixel**, where each array element stores

the coverage of the corresponding pixel location.

Output : The *Generalized Coverage Map.*

```
01. for i ← 0 to 1023 by 16 {
02.         for j ← 0 to 1023 by 16 {
03.                 Count ← 0
04.                 Max ← 0
05.                 for m ← 0 to 15 by 1 {
06.                         for n ← 0 to 15 by 1 {
07.                                 x ← i + m
08.                                 y ← j + n
09.                                 Count ← Count + coverage of Pixel[x][y] } }
10.                 MergedPixel[i][j] ← Count
11.                 If (MergedPixel[i][j]>Max)
12.                         Max ← MergedPixel[i][j] }}
13. for i ← 0 to 1023 by 1 {
14.         for j ← 0 to 1023 by 1 {
15.         GenCov[i][j] ← 255 × (1-MergedPixel[i/16][j/16]/Max)}}
```

In order to analyze and compare test results between the hierarchical and non-hierarchical schemes, we compute various forms of environment coverage statistics as follows:

- Over-Coverage ($TIOCP$)
- Coverage ($TICP$)

The actual computation of these statistics is discussed later. However, before we can discuss coverage, we first reviewed the vector-based navigation mechanisms from Chapter 3 and provide some test results that help to assess their individual usefulness.

4.2 Vector Comparison Experiments

Chapter 3 explained the mechanism used for implementing a *Global Average Position Vector* ($GAPV$) and a *Last Hundred Average Position Vector* ($LHAPV$). Recall that the *Global average positions* (GAP) represents the moving average of all previous leader positions. As time goes on, the GAP becomes closer to the centroid of the area covered. By setting up a $GAPV$ based on the GAP, the worker robots can be steered either toward or away from this centralized GAP location.

$LHAPV$ directs robots to make the navigation decision based only on recent local coverage (i.e. the average of the last 100 leader locations). $GAPV$ leads robots to new regions upon the more globally information. According to the task requests, one of them can be selected to direct HCSMRS to execute task. In order to compare the different effect that the two types of vectors make on the coverage, we first compared the traces generated by *Last hundred average position* and *Global average position.*

Recall as well that the *Last hundred average position* ($LHAP$) represents the average of the last 100 leader locations (i.e., a kind of "local average"). In the same way as the $GAPV$ was used, the $LHAPV$ can be set up based on the $LHAP$. The use of the $LHAPV$ allows the robots to steer away from (or toward) recent locations. For example, if robots become "stuck" within a cluster of obstacles, the $LHAPV$ can help robots get out from this difficult situation. Since the $LHAP$ quickly moves

toward the "stuck" locations, thereby providing incentive for the leader to move away from the cluster towards more open areas.

Before we comparing these two average positions, we first consider the *Out-of-Range Count Vector* shown in Figure 4.5. Consider the situation in which the leader robot is getting close to touching the environmental boundary. This makes the roaming space for robots R_{TL} and R_{BL} very limited. Therefore, the "Out-of-Range" counts for robots in both $Zone_{TL}$ and $Zone_{BL}$ become much larger than that in $Zone_{TR}$ and $Zone_{BR}$. According to the mechanism for calculating the $ORCV$, as time passes, the $ORCV$ will point away from the boundary since the likelihood that RTR and RBR robots will have higher "Out-of-Range" counts increases .

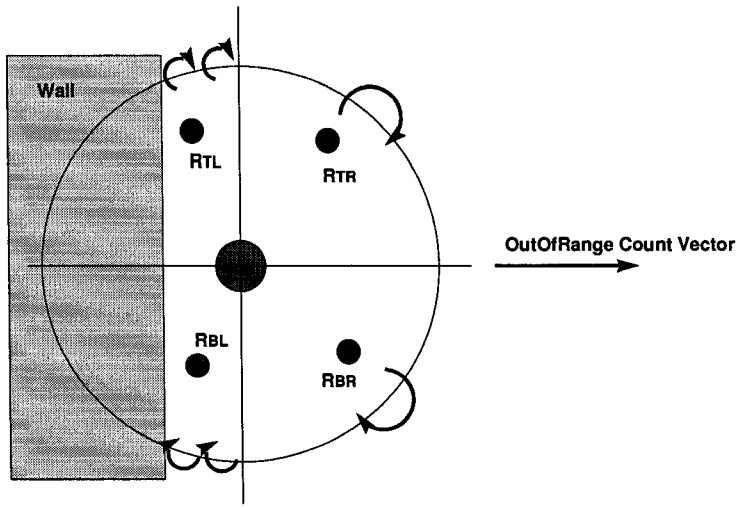


Figure 4.5: Example showing how the $ORCV$ avoids boundaries.

Consider a further scenario as shown in Figure 4.6, where we use the $ORCV$ (A) and not use the $ORCV$ (B). Notice how quickly the $LHAPV$ is able to "escape" from inside the obstacles, while the random movements took longer.

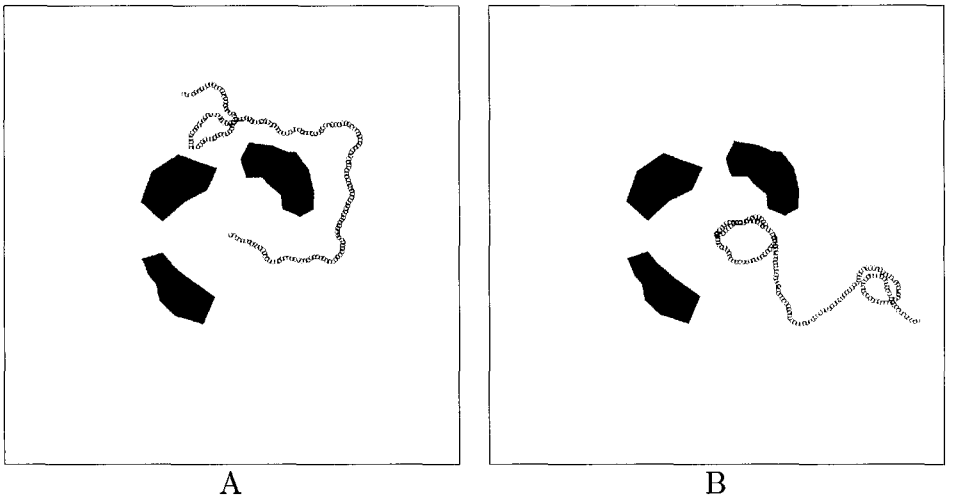


Figure 4.6: The leader's trace generated (A) the *Out-of-Range Count Vector* and (B) Random movement.

Figure 4.7 shows the coverage results with the leader trace through applying $ORCV$ and $LHAPV$ in an empty environment. Notice that the $ORCV$ tends to keep the robot away from the borders whereas the $LHAPV$, although keeping the robots moving, tends to cause the robots to "Rub up" against the boundary since it is trying to steer the robots away from a more central location.

When the GAP and/or $LHAPV$ are combined with the $ORCV$, the HCSMRS achieve better performance. For example, HCSMRS robots benefit from the combined vector $GAPORCV$, which not only helps them to expand out or shrink towards a region, but also helps them to avoid obstacles and boundaries. Additionally, by changing the ratio of the magnitudes of the two combined vectors, we can get an arbitrary tradeoff as desired. For example, we can enlarge the magnitude of the $GAPV$ when applying $GAPORCV$ if we want robots to keep away from a central area. On the other hand, if we let the magnitude of $ORCV$ become larger than that

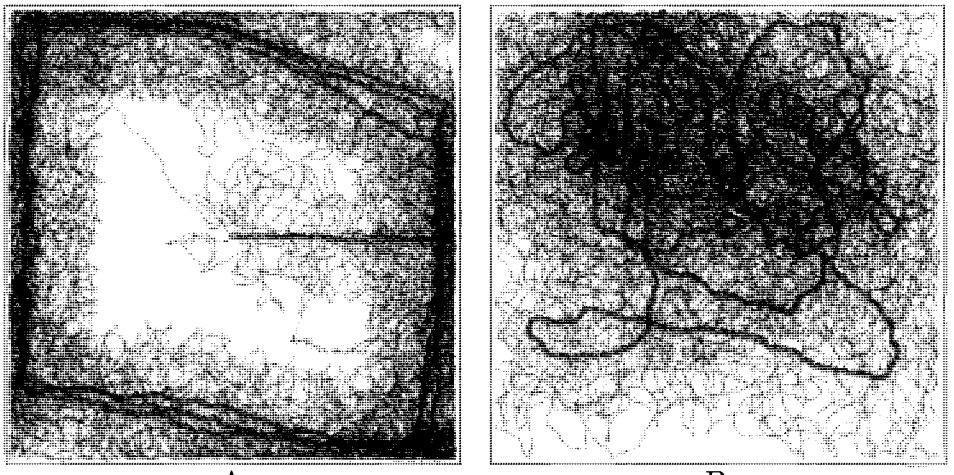


Figure 4.7: *Location Coverage* made by (A) the $LHAPV$ and (B) the $ORCV$.

of the $GAPV$, robots then pay more attention to the central area. The magnitude ratio can be defined based on Equation 4.1

$$Ratio_{Mag} = Mag_{GAP}/Mag_{ORCV} \tag{4.1}$$

A similar ratio can be defined by combining the $LHAPV$ and $ORCV$ to obtain a $LHAPORCV$. Consider comparing the $GAPORCV$ and $LHAPORCV$ by first analyzing average position traces, as shown in Figure 4.8. Notice that when $ORCV$ has larger magnitude than the $GAP/LHAPV$ are more clustered towards a central location. On the other hand, when the $ORCV$ decreases in magnitude, the robots tend to expand outward. Also, when comparing the $LHAPV$, its path tends to smooth towards an outer loop shape when the $ORCV$ is reduced, since the $ORCV$'s influence to push away from the boundaries is lessened.

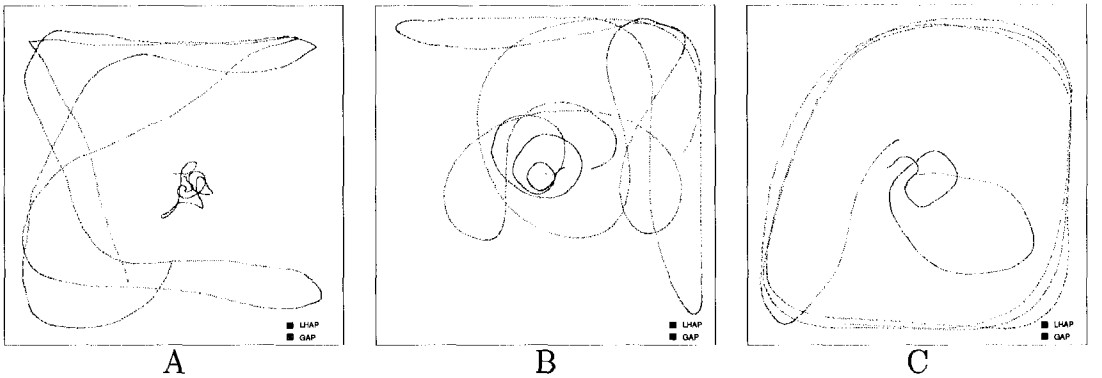


Figure 4.8: *GAP* trace and *LHAP* trace with three different $Ratio_{Mag}$ (A) $Ratio_{Mag} = 1/2$, (B) $Ratio_{Mag} = 1/1$ and (C) $Ratio_{Mag} = 1/0.5$.

Figure 4.9(A) shows the *Location Coverage* and leader trace for the *LHAPORCV* test corresponding to Figure 4.8B. Notice how the leader closely the *LHAP* (since the *LHAP* is just a moving average of the leader's positions). The top right border rubbing is clearly evident, yet not as pronounced as in Figure 4.7A. Clearly, the result is an equal tradeoff between the coverage of Figure 4.7A and 4.7B. Similarly, Figure 4.9(B) represents a similar tradeoff between the *GAPV* and *ORCV*.

4.3 Centralized Concentration Coverage

As shown in Figure 4.8, the *GAP* trace expands outwards as the magnitudes of the *ORCV* decrease. Figure 4.10 displays three corresponding *Location Coverage* and leader traces by applying the vector combined *GAPORCV* with different ratios. With different $Ratio_{Mag}$, the leader generates a trace with different outward-expanding radii.. The larger the $Ratio_{Mag}$, the bigger the radius of the trace.

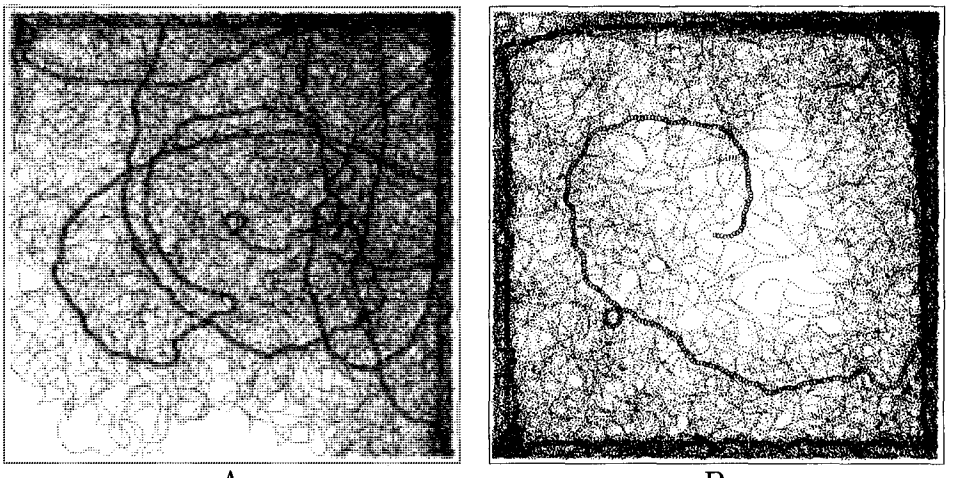


Figure 4.9: *Location Coverage* and the leader's trace after applying the two different combined vectors, (A) by *LHAPORCV* (B) by *GAPORCV*.

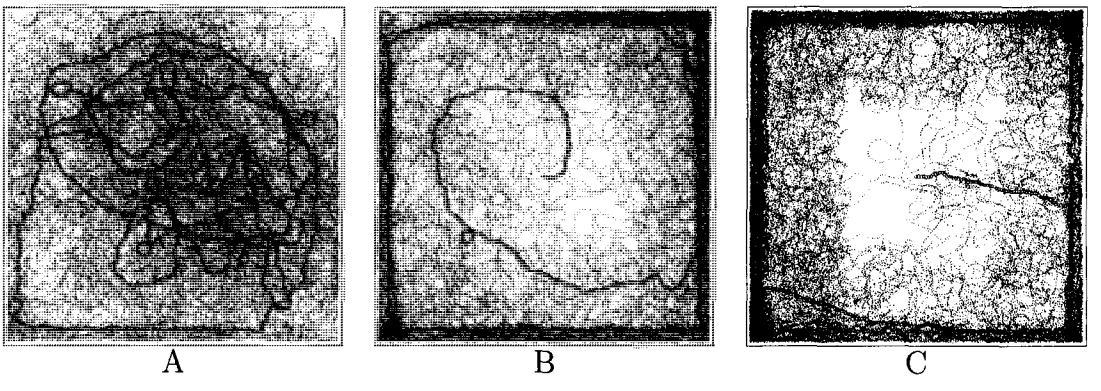


Figure 4.10: *Location coverage* and the leader's trace from applying the combined *GAPORCV* using different ratios, (A) $Ratio_{Mag} = 1/2$, (B) $Ratio_{Mag} = 1/1$ and (C) $Ratio_{Mag} = 1/0.5$.

As discussed in Chapter 3, the *GAPV* keeps robots "away" from a central area, while the *OGAPV* keeps robots "within" a central area. By using different values for $Ratio_{Mag}$, the leader can focus on different ranges. Figure 4.11 shows the *Location Coverage* and leader traces after applying *OGAPVORCV* (i.e., the combination of *OGAPV* and *ORCV*) with three unique ratios.

In order to investigate the effects that obstacles have on our coverage, we also performed a test using 1/1 *OGAPVORCV* ratio in an obstacle-cluttered environment of Figure 4.1A. The results are shown in Figure 4.12. Notice how the coverage is similar to that of Figure 4.11C.

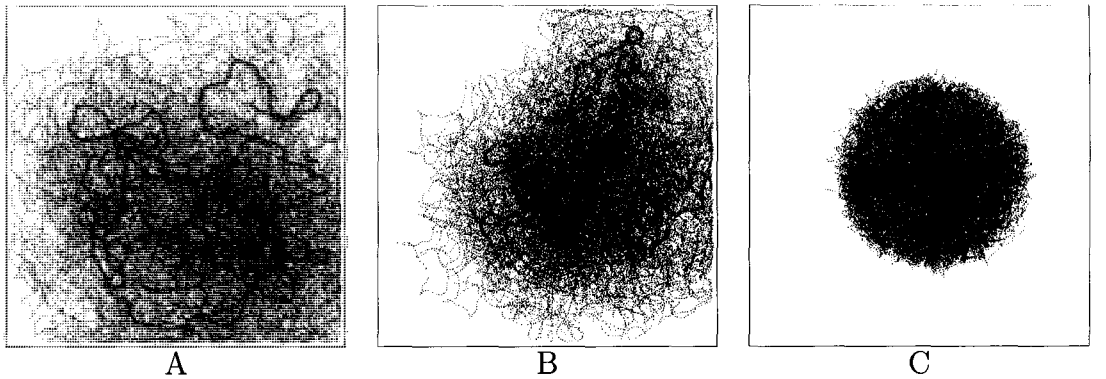


Figure 4.11: *Location Coverage* and the leader's trace from applying the combined *OGAPVORCV* using different ratios, (A) $Ratio_{Mag} = 1/2$, (B) $Ratio_{Mag} = 1/1$ and (C) $Ratio_{Mag} = 1/0.5$.

One such application of this *OGAPORCV* is that of producing outward-expanding ring-like search patterns. This can be achieved by providing dynamically changing ratios. We performed such a test by alternating the $Ratio_{Mag}$ at 4 unique times $Time_0$, $Time_{5,000}$, $Time_{10,000}$ and $Time_{15,000}$ during the test run. We started with

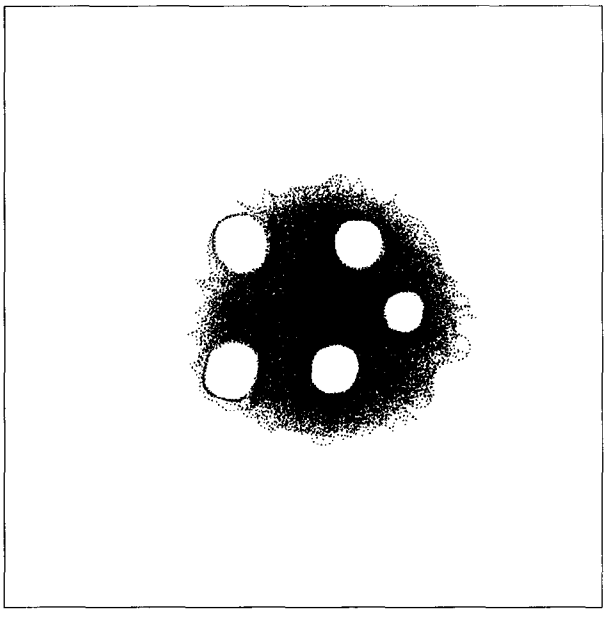


Figure 4.12: *Location Coverage* and the leader's trace from applying the combined $OGAPVORCV$ using ratio $Ratio_{Mag} = 1/0.5$ in an obstacle-cluttered environment.

a ratio of 1/0.5 and then altered it to 1/1, 1/1.5 and 1/2 at these times along the way. Figure 4.13 shows the colored *Generalized Coverage Map* for this test. Each color shown in this figure represents the robots' coverage within that producing time range. The results show the ability of the *OGAPORCV* to produce an outwards-expanding centralized search pattern, allowing worker robots to concentrate their efforts to produce a thorough coverage within a specific distance from the central location.

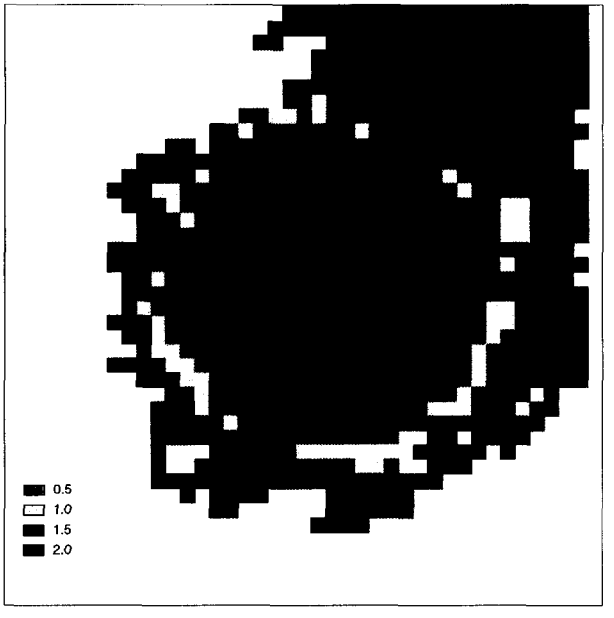


Figure 4.13: Colored *Generalized Coverage Map* for a search pattern strategy.

To take a closer look at the coverage produced by this test, consider partitioning the environment into four ring-like zones, $Zone_0$, $Zone_1$, $Zone_2$ and $Zone_3$, as shown in Figure 4.14. After each 5000 time steps, the ratio is altered to allowing the worker robots to expand outwards to the next zone.

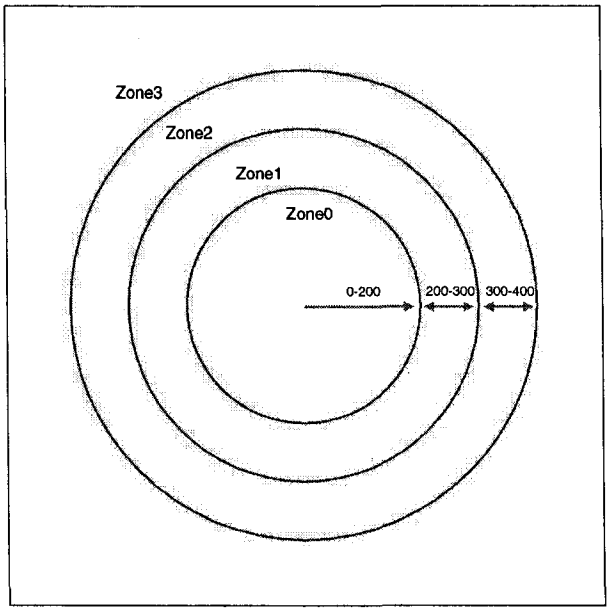


Figure 4.14: Ring-like environment zone partition.

The relationship between time ranges and the four zones has been shown in Table 4.3. In this table, CP_{ij} denotes the percentage of the covered points made within $Zone_j$ during the corresponding time range, $Range_i$.

Time Range	$Zone_0$	$Zone_1$	$Zone_2$	$Zone_3$
$0 - 4999$	CP_{00}	CP_{01}	CP_{02}	CP_{03}
$5000 - 9999$	CP_{10}	CP_{11}	CP_{12}	CP_{13}
$10000 - 14999$	CP_{20}	CP_{21}	CP_{22}	CP_{23}
$15000 - 20000$	CP_{30}	CP_{31}	CP_{32}	CP_{33}

Table 4.1: Corresponding relationship between four time ranges and the four zones.

CP_{ij} can be calculated as follows:

$$CP_{ij} = Number\ of\ pixels\ reached\ within\ Zone_j\ during\ Range_i$$
$$/Number\ of\ pixels\ reached \quad (4.2)$$

Since each of our worker robots runs 20,000 time units per test, the number of pixels that all four worker robots reached will be 80,000. We compared the per zone coverage for our HCSMRS scheme vs. the non-hierarchical scheme. For the non-hierarchical scheme, the 4 worker robots moved randomly, hence not focused on particular zones at any particular time. Figure 4.15 shows the *Location coverage* of with the leader's trace for the HCSMRS and non-hierarchical schemes. As we can see, the non-hierarchical MRS generates an uniformed *Location coverage*, while the HCSMRS produces a more centralized coverage.

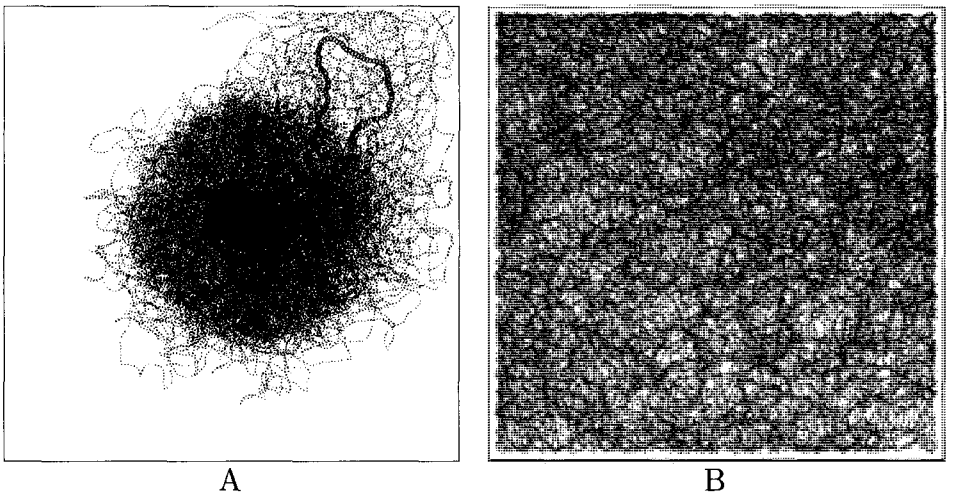


Figure 4.15: *Location Coverage* and the leader's trace made by (A) *Centralized Concentration Coverage* and (B) a non-hierarchical MRS.

Figure 4.16 shows the coverage for the HCSMRS test. Notice that the coverage is high in $Zone_0$ for the initial time range and gradually this amount becomes distributed across the other zones over time. In contrast, Figure 4.17 shows the results from the non-hierarchical scheme. In this case, the robots consistently spend most time outside $Zone_0$ and in the outer $Zone_3$ because $Zone_3$ is much larger than $Zone_0$ and the robots are moving uniformly throughout the environment. There is no ring-like coverage at all. Hence, the HCSMRS clearly provides a more focused coverage in a flexible manner.

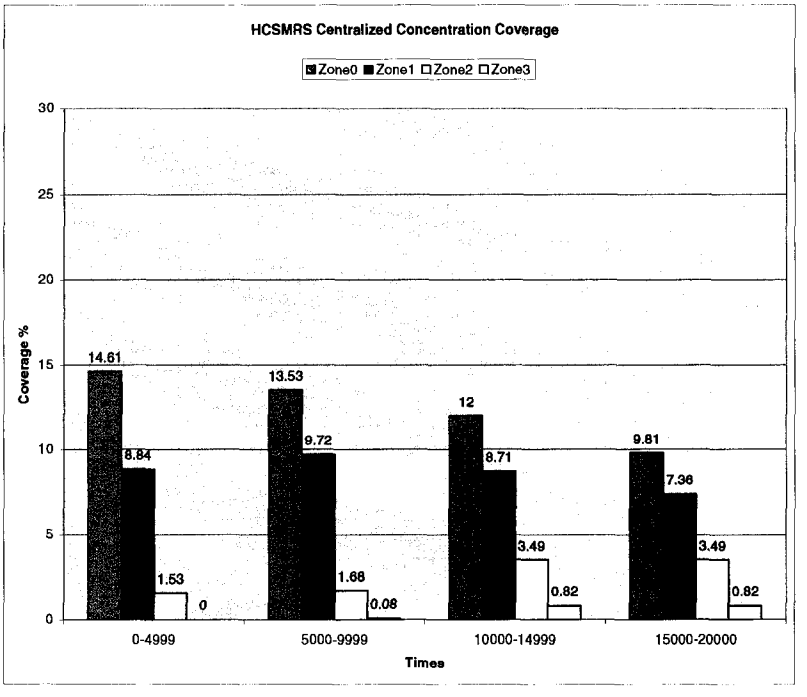


Figure 4.16: Coverage for the HCSMRS *Centralized Concentration Coverage* test.

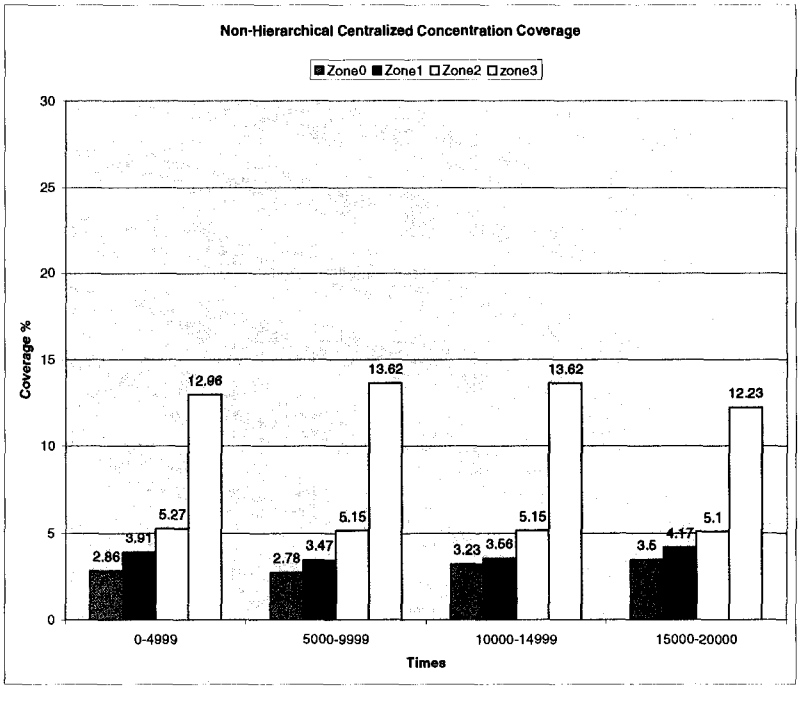


Figure 4.17: Coverage for the non-hierarchical *Centralized Concentration Coverage* test.

4.4 Path Concentrated Coverage

In Chapter 3, we discussed networks that can be used to perform $Point - to - Point$ navigation. Using these networks, it is possible to attain *Path-Centralized Coverage*. In this test, we let the robots start from a central point and continuously follow a pre-determined path, $Goal_0 \rightarrow Goal_1 \rightarrow \cdots \rightarrow Goal_7$, within environments with and without scattered obstacles (see Figure 4.1B). Figure 4.18 shows the leader's trace in both environments. As we can see, the leader's trace shown in Figure 4.18A has more curved parts and traverses a larger area. The trace in Figure 4.18B is straighter and follows more closely the desired path.

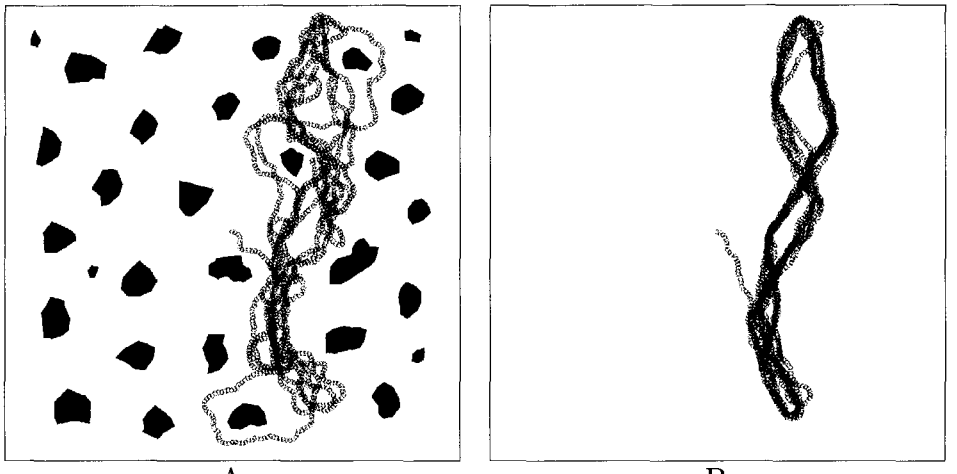


Figure 4.18: The leader's trace in (A) a non-empty and (B)an empty environment.

Figure 4.19 shows the *Generalized Coverage Map* made by both the HCSMRS robots and the non-hierarchical robots in empty and non-empty environments.

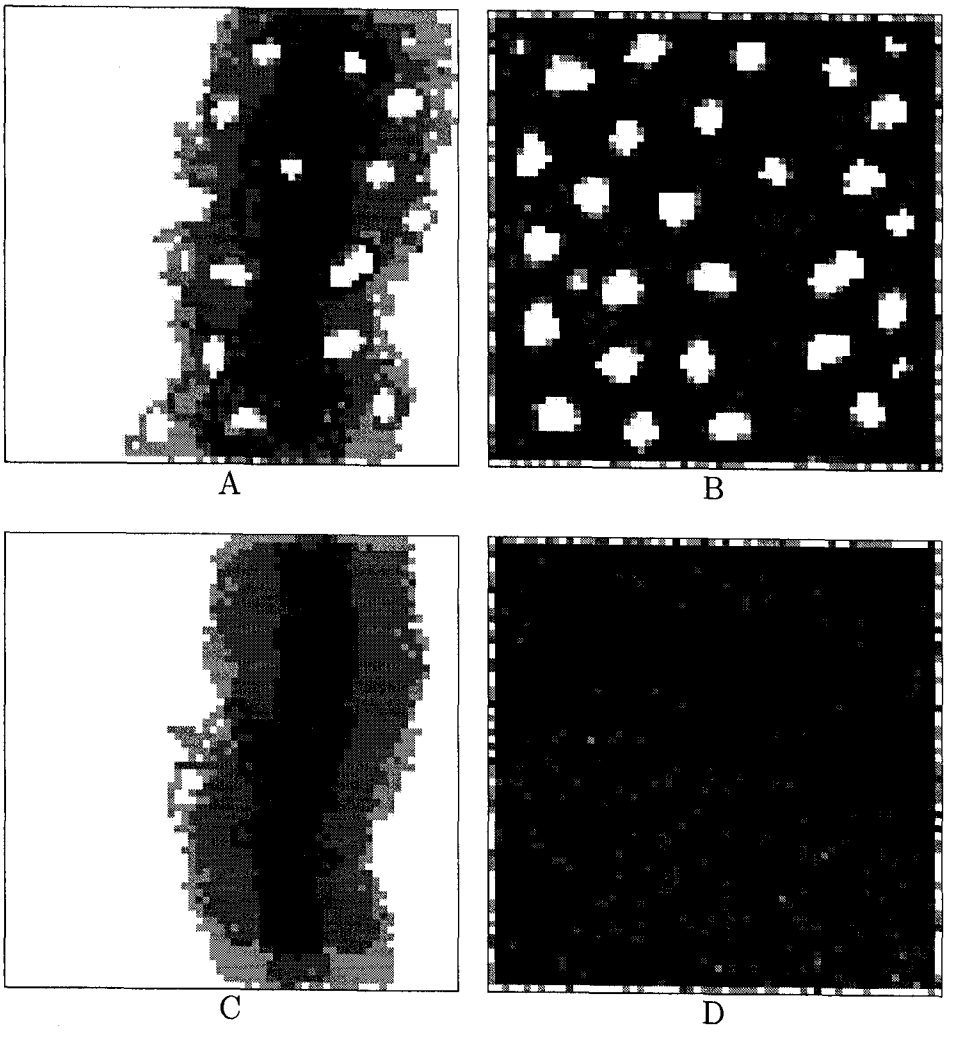


Figure 4.19: The *Generalized Coverage Map* made by HCSMRS in both (A)a non-empty environment and (C)an empty environment. And *Generalized Coverage Map* made by non-hierarchical MRS in both (B) a non-empty environment and (D)an empty environment.

To analyze the coverage, we define two notations, namely, *Actual Coverage* (*AC*) and the *Approximate Coverage* (*PC*). *Approximate Coverage* is defined as the area that is supposed to be covered by the robots theoretically. Usually, to generate the *PC*, we consider an ideal path that the leader robot should follow, and consider the area along that path as the *PC* (see Figure 4.20A). Since *Actual Coverage* is defined as the actual area covered by worker robots, the *AC* is produced according to the real path that the leader traveled along the path (see Figure 4.20B). By comparing the *AC* made by both the HCSMRS and the non-hierarchical MRS with the *PC*, the advantage that the HCSMRS has respect to *Path-Concentrated Coverage* can be verified.

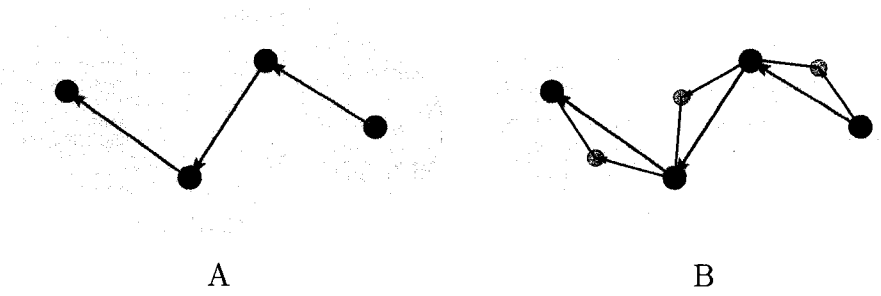


Figure 4.20: A: Example of (A) the *Approximate Coverage* and (B) the *Actual Coverage*.

Figure 4.21 shows the overlap produced by the *AC* and *PC* areas for our test cases for empty and obstacle-cluttered environments. The *AC* is always a larger area since the leader never moves precisely along the desired path. Notice that the presence of obstacles further accentuates the overlap area differences.

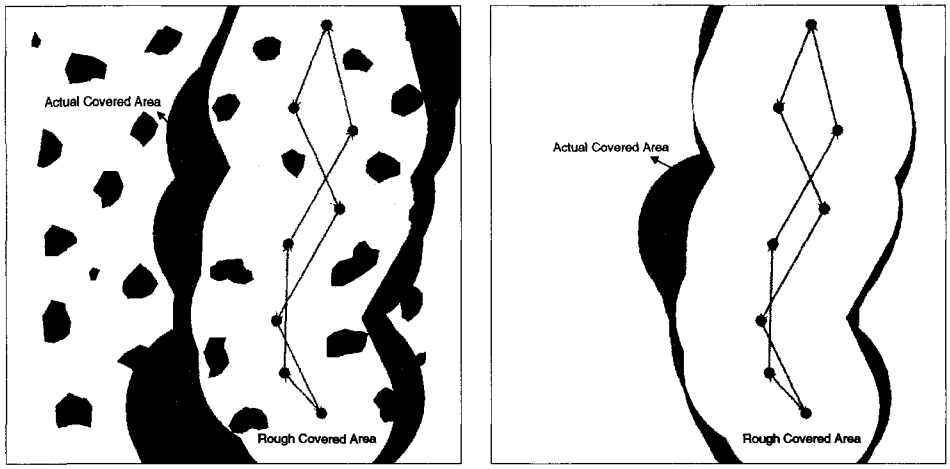


Figure 4.21: Examples of the coverage areas made by the HCSMRS, in (A) an obstacle-cluttered environment and (B) an empty environment.

To show the difference in AC and PC produced by the HCSMRS and the non-hierarchical scheme in the same test case, we calculate three values, representing the coverage statistics as follows.

- **Over-coverage concentrated an area**

$$Over - Coverage = Number\ of\ pixels\ reached\ more\ than\ once\ within\ the\ area / Number\ of\ pixels\ reached \quad (4.3)$$

- **Coverage concentrated over an area**

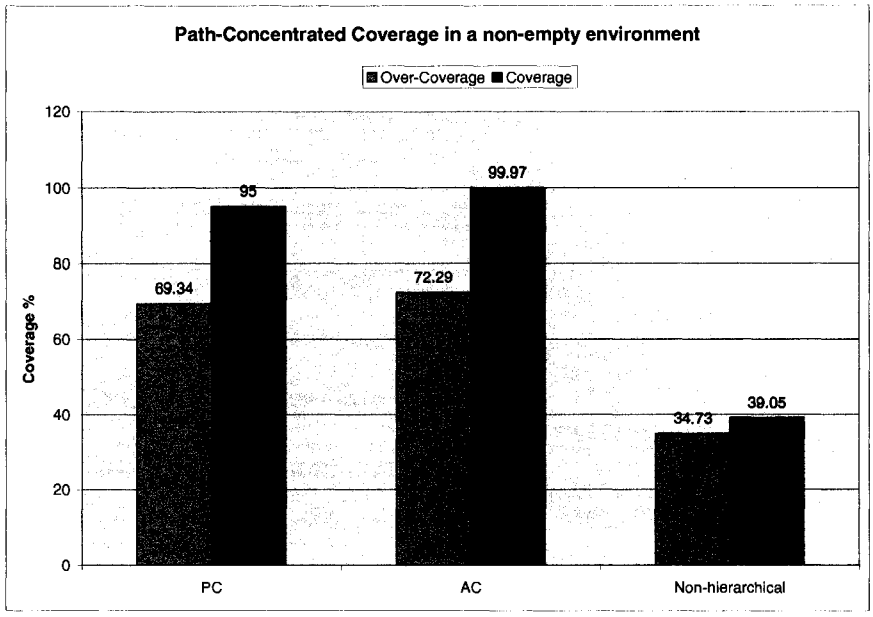
$$Coverage = Number\ of\ pixels\ reached\ within\ the\ area \\ /Number\ of\ pixels\ reached \quad (4.4)$$

Figure 4.22 shows these three values made by HCSMRS and the non-hierarchical scheme MRS. The three values of PC, $Non-hierarchical$ in both non-empty (Figure 4.22A) and empty (Figure 4.22B) environments are much less than that made by HCSMRS. This demonstrated that HCSMRS has much higher concentration on the path to be followed.

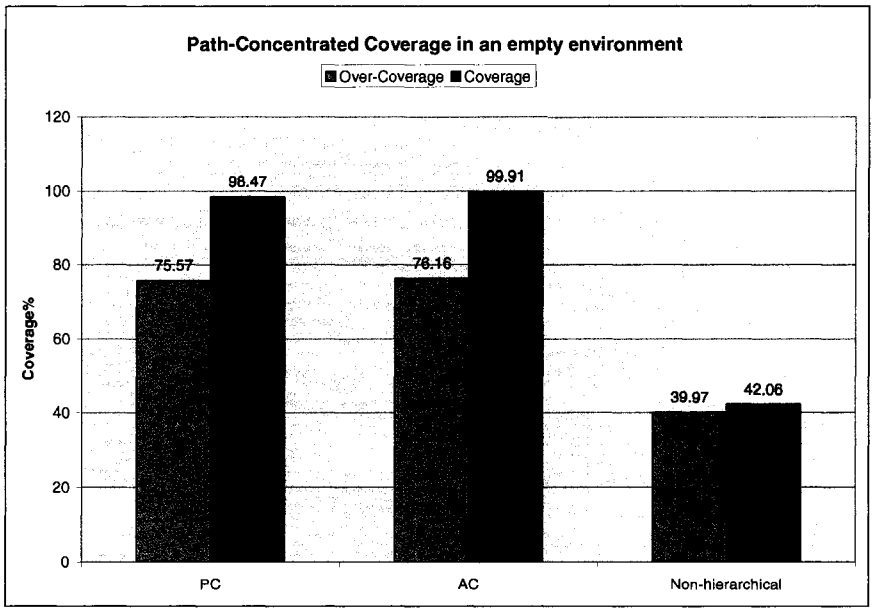
4.5 Conclusion

In this Chapter, we set up test scenarios for testing several vector based navigation patterns, such as $GAPV$, $LHAPV$ and $ORCV$, or the vector combination of two of them. The $ORCV$ has been demonstrated to be capable of keeping robots away from areas with obstacles as well as boundaries. The $LHAPV$ also keeps robots away from obstacles, but has worse performance if avoiding boundaries is important. The $GAPV$ has been shown that it works well on *Ring-Like search patterns*, through justifying the $Ratio_{Mag}$. In order to get a better performance, we compared the combined vectors, $GAPORCV$ and $LHAPORCV$ under different $Ratio_{Mag}$.

Since the non-hierarchical MRS has no means of directing its workers, it was unable to provide adequate coverage for scenarios that demand concentrated coverage. Through our tests, we were able to show that in both empty and non-empty environments, the HCSMRS strategy provides better over-coverage along any path as well as over various centralized areas.



A



B

Figure 4.22: Coverage for the HCSMRS *Path-Concentrated Coverage* test in (A) a non-empty environment and (B) an empty environment.

Chapter 5

Conclusion

In this thesis, we introduced a Hierarchical Communication Architecture for Multi-Robot Systems (*HCSMRS*). As shown in the model of Figure 1.1A, the HCSMRS is a layered structure where each layer consists of small local communication groups. Each communication group is composed of a number of worker robots and a leader robot. The worker robots can only communicate with their leader directly. Workers can communicate with other worker robots within the same communication group by relaying messages through their shared leader. A leader robot directs its worker robots within its communication range and is responsible for making decisions for the whole group. In a multi-layer hierarchy, the upper layer worker robots are the leaders for the lower layer communication groups.

In a HCSMRS, the top-most leader maintains the global information of the environment, the assessment of task progress and completion, the status of lower level robots and the overall coordination among different communication groups. When performing a large and complex task, it is necessary for the top-most leader to send the corresponding knowledge down to individual communication groups when assigning

and switching tasks. For example, when performing mapping, information pertaining to coverage, area distribution and robot failure must be reported to the top-most leader from lower level groups occasionally. The top-most leader then makes decisions and begins to re-assign and re-organize the groups in order to more efficiently complete the mapping. Via this hierarchical architecture, the entire multi-robot system can be coordinated properly and orderly, which should guarantee a more efficient task completion than with a non-hierarchical approach.

As shown in the framework of in Figure 1.9, the entire system is composed of the basic *movement-related*, *communication-related* and *navigation-related* behaviors and mechanisms. In this thesis, we chose a bottom-up behavior-based approach to implement all control behaviors and mechanisms, because it generally results in a more reactive system with quicker response time. Each behavior was implemented through "Hardwired" neuron networks in order to provide fast flow-related control and reduce the need for high computational power. We presented both fixed and dynamic PTP navigation mechanisms and behaviors (i.e., Vector-based navigation scheme, Path-Concentrated navigation scheme) that allows the MRS to focus on particular areas of the environment.

Although the amount of previous work on hierarchical communication architectures for robots has increased recently, most of the work seems to have focused on pattern formation ([5][7][13][14][18][19][20][24][25][1][26][27][28][29][15][22][6][21]), leader selection ([7][19]), group selection ([19]) and inter-group communications ([13][14][18] [24][25]). Our work differs from the previous work in that we ignore cooperation aspects among different communication groups and we do not analyze the "Quality" of various hierarchical structures. Instead, we focus on coordinating our hierarchical group of robots to achieve a particular area coverage through the implementation of

various navigation mechanisms.

We implemented the necessary behaviors and mechanisms required for building up a two layer HCSMRS. During the course of implementation, a few issues arose in regards to our communication and navigation strategies.

The first issue was in regard to our "Out-of-Range" avoidance behavior. The "combined vector" solution that we adopted does not guarantee that a worker robot will always remain within communication range. As shown in the example of Figure 2.24, large obstacles may prevent a worker robot from re-gaining safe communication with its leader. However, we found that by reducing the leader's speed significantly, with respect to the worker's speed, this potential problem did not surface.

One other observed situation was that of a robot becoming trapped in a "local minima" situation, possibly becoming stuck. If the leader were to move away while a worker is stuck, the worker could become last (i.e., left behind). We found that the use of random movements in the wandering behavior was able to free the robot from "stuck" positions and in our experiments, no worker robot ever become lost.

Another issue came to the fore during the implementation of the LHAPV-based network. In order to store an array of 100 pervious positions, a very large network is needed, although it is composed of 100 smaller network positions that are near identical. We did not implement this network, but instead used a hard-coded array to simulate it, since our purpose was simply to investigate the effects of maintaining a running average.

A third issue that we did not address was related to our path-directed coverage. We did not investigate where such paths came from. That is, such paths would likely be specified by the leader and high level paths would come from the top-most layer in the hierarchy. Such paths depend on the application, but we did not attempt, in this thesis, to apply any particular application to our HCSMRS model. Therefore, we simply hard-coded the paths and did not address issues pertaining to dynamic path selection.

Through experimentation, we aimed to demonstrate how our HCSMRS can provide advantages over non-hierarchical multi-robot systems. The HCSMRS does not necessary provide advantages over non-hierarchical approaches for all applications. It is likely that the true advantages will depend on the complexity of the task being performed. Since our focus was on setting up and investigating the low-level communication and navigation mechanisms, we did not investigate various applications. Instead, we decided to analyze the coverage of the HCSMRS as the path of the system focussed over particular areas of the environment. We tested a two layered HCSMRS under two test scenarios, *Path-Concentrated Coverage* (see Figure 1.2A) and *Centralized Concentration Coverage* (see Figure 1.2B). There was no need to analyze a multi-level hierarchy since any advantages or disadvantages of such a hierarchy are somewhat meaningless without a particular application in mind. By comparing the test results with those of a non-hierarchical strategy, we were able to show that our approach obtains better centralized and path-directed coverage, highlighting the ability of the HCSMRS to focus on particular regions.

For the *Path-Concentrated Coverage* test, we compared the *Over Coverage* and *Coverage* with regards to a fixed path made by HCSMRS in both empty and non-empty environments with the coverage achieved by a non-hierarchical scheme (see

Section 4.4). As shown in Figures 4.22A and 4.22B, both the *Coverage* and *Over Coverage* made by the HCSMRS is much much higher than those made by non-hierarchical scheme. For the *Centralized Concentration Coverage* test, we similarly compared the *Coverage* within each of the four worker zones within a unique time range for both the HCSMRS and the non-hierarchical schemes (see Figures 4.16 and 4.17). Again, the results showed a better (i.e., more focused) coverage within the centralized area, which can be advantageous for solving outwards-expanding search-related problems.

5.1 Future Work

Although neuron networks are reasonable for implementing behaviors and mechanisms, there is room for improvement. For example, as mentioned in Chapter 3, we discussed the difficulty of implementing the *Last Hundred Average Position Vector* network since it would need to be quite large to store the 100 previous locations. In the future, it would be wise to develop such a fully-expanded neuron network to verify its feasibility and practicality. However, we did not find that this LHAPV provided any noticeable advantage of the *Global Average Vector* network. In one particular random test (not discussed in the thesis), our worker robots left their "safe" communication range a total of 26.4% of the time. However, during our tests, no robot became lost by leaving the critical communication range at any time. A further investigation would need to be done in order to determine the factors that might cause the robots to lose contact with their leader (e.g., in environments with large, complex obstacles, or when the leader's communication range decreases). Experiments should be performed to determine the effects of reducing/increasing the safe and critical communication ranges and how these can be adjusted dynamically. Furthermore, we

hope to develop "saving" strategies to steer the group towards locations where worker robots became lost.

Another avenue for future study is in regards to testing our hierarchy with multiple layers. We need to examine the interference/overlapping effects that two separate communication groups may have. Also, we can investigate varying the sizes of the communication groups and perhaps allow "lost" robots to be "picked-up" by a different group of robots so as to continue its task work.

This thesis was aimed at coordinating worker robots so that the remain in communication range with their leader robot at all times, while minimizing the overlap with other worker robots in the same group. As such, we did not investigate any specific tasks that the workers could perform. That is, our worker robots merely moved randomly, while steering back to the leader or to its zone in order to maintain proper balanced coverage. In the future, we need to examine the performance of the HCSMRS when performing various tasks such as searching, mapping, foraging etc.. This opens up other areas of investigation related to task distribution and fault tolerance within the hierarchy. Also, it would be nice to investigate the hierarchy's performance in dynamically changing environments.

This work represents a small step towards understanding the complex issues involved with coordinating groups of robots. We hope to have shown that simple, low-level mechanisms (i.e., neuron networks) can be used successfully to coordinate multiple groups of robots in an efficient and organized manner. The approach here just represents the coordination and communication framework for more complex task-oriented multi-robot systems. It is our hope that research continues in this area so that robust, reliable, efficient multi-robot systems can be used to solve difficult

tasks in a timely manner.

Bibliography

[1] R. W. Beard, J. Lawton and F. Y. Hadaegh, "A coordination architecture for spacecraft formation control", *IEEE Transactions on Control Systems Technology*, vol. 9, Issue 6, pp. 777-790, Nov. 2001.

[2] R. D. Beer, "Intelligence as Adaptive Behavior", *Perspectives inArtificial Intelligence*, vol. 6 Academic Press, 1990.

[3] R. Brook, "Intelligence without representation", *Artificial Intelligence*, vol. 47, pp.139-159, 1991.

[4] Y. U. CAO, A. S. Fukunaga and A. B. Kahng, "Cooperative Mobile Robotics: Antecedents and Directions", *IEEE/TSJ International Conference onIntelligent Robots and Systems*, vol. 4, pp. 7-23, March, 1997.

[5] R. Damoto, A. Kawakami and S. Hirose, "Study of Super-Mechano Colony: concept and basic experimental set-up", *Advanced Robotics*, vol. 15, Issue 4, pp. 391-408, 2001.

[6] J. P. Desai and J. Ostrowski and V. Kumar, "Controlling Formations of Multiple Mobile Robots", *IEEE International Conference on Robotics and Automation*, pp. 2864-2869, May 1998, Leuven, Belgium.

[7] M. B. Dias and A. Stentz, "Opportunistic optimization for market-based multirobot control", *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2714-2720, 2002.

[8] J. M. Egerstedt and X. Hu. "Formation constrained multi-agent control", *IEEE Transactions on Robotics and Automation*, vol. 17, Part 6, pp. 947-951, 2001.

[9] N. E. Leonard and E. Fiorelli, "Virtual Leaders, artificial potentials and coordinated control of groups", *Proceedings of the 40th IEEE Conference on Decision and Control*, pp. 2968-2973, Dec. 2001, Orlando, FL.

[10] R. Fierro, A. K. Das, V. Kumar and J. P. Ostrowski, "Hybrid control of formations of Robots", *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 157-162, May, 2001, Seoul, Korea.

[11] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "Collaborative multi-robot localization", In *Proceedings of the German Conference on Artificial Intelligence and the 21st Symposium on Pattern Recognition*, pp. 255-266, 1999.

[12] L. Iocchi, D. Nardi and M. Salerno, "Reactivity and Deliberation: A Survey on Multi-Robot Systems", *Lecture Notes in Computer Science*, vol. 2103, pp.9, 2001.

[13] M. Ji, M. Egerstedt, G. Ferrari-Trecate and A. Buffa, "Hierarchical containment control in heterogeneous mobile networks", Proc. *Mathematical Theory of Networks and Systems*, pp. 2227–2231, 2006, Kyoto, Japan.

[14] M. Ji, A. Muhammad and M. Egerstedt, "Leader-based multi-agent coordination: controllability and optimal control", *American Control Conference*, pp. 6, 2006.

[15] V. Kapila, A. G. Sparks, J. M. Buffington and Q. Yan, "Spacecraft formation flying: Dynamics and control", *Journal of Guidance, Control and Dynamics*, vol. 23, pp. 561-564, May-June, 2000.

[16] M. Lanthier and F. Oppacher, "Towards the Design of Smaller Artificially Living Robots", *Intelligent Autonomous Systems 4 (IAS-4)*, IOS Press, pp. 687-693, 1995, Karslruhe, Germany.

[17] M. Lanthier, "Implementation of Adaptive Behaviours in a Simple Insect-Like Robot", *Master's thesis*, School of Computer Science, Carleton University, Ottawa, 1993.

[18] S. Li, L. Shuai, X. Cheng, Z. Tang and J. Yang, "A Descriptive Model of Robot Team and the Dynamic Evolution of Robot Team Cooperation", *International Journal of Advanced Robotics Systems*, vol.2, pp. 139-143, 2005.

[19] D. Miller, "Multiple behavior-controlled microrobots for planetary surface missions", *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, pp. 289–292, 1990, Los Angeles, USA.

[20] F. R. Noreils, "Toward a Robot Architecture Integrating Cooperation between Mobile Robots: Application to Indoor Environment", *International Journal of Robotics Research*, vol. 12, no. 1, pp. 79-98, 1993.

[21] S. Sheikholeslam and C. A. Desoer, "Control of interconnected nonlinear dynamical systems: The platoon problem", *IEEE Transactions on Automatic Control*, vol. 37, pp. 806-810, June, 1992.

[22] T. Sugar and V. Kumar "Decentralized control of cooperating mobile manipulators", *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2916-2921, May, 1998, Leuven, Belgium.

[23] P. Tabuada, G. J. Pappas and P. Lima, "Feasible formations of multi-agent systems", *Proceedings of the American Control Conference*, pp. 56-61, June 2001, Arlington, VA.

[24] H. G. Tanner, G. J. Pappas and V. Kumar, "Leader-to-formation stability", *IEEE International Conference on Robotics and Automation*, pp. 443-455, 2003.

[25] H. G. Tanner, V. Kumar and G. J. Pappas, "The effect of feedback and feedforward on formation ISS", *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 3448-3453, 2002, Washington, DC.

[26] P. K. C. Wang, "Navigation Strategies for multiple mobile robots moving in formation", *Proceedings of IEEE International Workshop on Intelligent Robots and Systems*, pp. 486-493, Sept. 4-6, 1989.

[27] P. K. C. Wang and F. Y. Hadaegh, "Coordination and control of multiple microspacecraft moving in formation", *The Journal of the Astronautical Sciences*, vol.44, no. 3, pp. 315-355, 1996.

[28] P. Wang, F. Hadaegh and K. Lau, "Synchronized formation rotation and attitude control of multiple free flying spacecraft", *AIAA Journal of Guidance, Control and Dynamics*, vol.22, pp.28-35, Jan. 1999.

[29] Q. Yan, G. Yang, V. Kapila and M. S. de Queiroz, "Nonlinear dyanamics and adaptive control of multiple spacecraft in periodic relative orbits", *23rd Annual AAS Guidance and Control Conference*, no. 00-013, Feb. 2000, Breckenridge, CO.

List of Abbreviations and Symbols Used

AC	Actual Coverage
CV	The combination of both *ORCV* and *GAPV* or *LHAPV*
D_R	The vector from the worker robot to the leader robot
D_{VC}	The desired direction of the combined vector
FPTP	Fixed Point-to-Point
GAP	The Global Average Position
GAPORCV	The combination of both *ORCV* and *GAPV*
GAPV	The Global Average Position Vector
HCSMRS	Hierarchical Communication Structure for Multi-Robot Systems
LHAP	Last Hundred Average Position
LHAPORCV	The combination of both *ORCV* and *LHAPV*
LHAPV	Last Hundred Average Position Vector
MRS	Multi-Robot System
OGAPORCV	The combination of *OGAPV* and *ORCV*
OGAPV	The Opposite Global Average Position Vector
ORCV	Out-of-Range Count Vector
PC	Approximate Coverage

PKN	Previous Knowledge based Navigation
PPV	Previous Position Vector
PTP	Point-to-Point
V_C	The combined vector
V_G	The vector from the worker robot to the last "good location
V_L	The vector from the worker robot to the leader's location